Control de la evolución de un robot móvil con visión estereoscópica

Autores

David Carracedo Díaz

Jorge Ibarra Mollá

Mario González Eiroa

Director:

José Jaime Ruz Ortiz

Curso: 2010/2011

Proyecto de Sistemas informáticos

Facultad de Informática

Universidad Complutense de Madrid

Autorización

Autorizamos a la Universidad Complutense de Madrid a utilizar y/o difundir con fines académicos y no comerciales, siempre mencionando expresamente a sus autores, tanto la propia memoria como el código, la documentación y/o el prototipo desarrollado.

Madrid, a 1 de Julio de 2011

David Carracedo Díaz

Jorge Ibarra Mollá

Mario González Eiroa

Resumen

El ser humano siempre ha estado buscando la forma de crear seres a su semejanza y que puedan realizar los trabajos más complicados o las tareas cotidianas. Con el paso del tiempo se ha conseguido que dichos seres, conocidos como robots, sean fácilmente utilizables e incluso programables, consiguiendo así, que un mismo robot se adapte a distintas necesidades.

Hoy en día, tenemos en el mercado multitud de robots que podemos adquirir y programar para adaptar a nuestras tareas. Sin embargo, la forma de programarlos no es accesible a todos, hay que tener una preparación adecuada para entender cómo realizar la programación correctamente.

El objetivo de este proyecto es crear un lenguaje fácil e intuitivo para controlar el robot Surveyor SRV-1 con comandos simples que controlen los movimientos del robot e interactúen con el usuario o con el entorno. Los programas de dicho lenguaje serán escritos en archivos XML debido a que existen abundantes recursos para procesar el lenguaje.

Además, hemos desarrollado un simulador del robot para ver el comportamiento de los programas escritos sin necesidad de disponer del robot real.

Para ambos casos, simulador o robot real, se podrán escribir programas editando archivos XML o mediante una interfaz que requiere mínimos conocimientos informáticos.

En esta memoria se describirá tanto la especificación del lenguaje como las diferentes tecnologías integradas en el proyecto, tales como la visión estereoscópica o la comunicación con el robot mediante voz.

Palabras clave: robot móvil, lenguaje de comandos, simulador 3D, visión estereoscópica, programación de robots.

Abstract

The human beings have always been looking for many ways to create beings in his likeness that were able to perform the most difficult jobs or the daily tasks. Overtime humanity has made these things, known as robots, easily usable and even programmable, to the point that the robot itself adapts to different needs.

Today there are a lot of different robots we can acquire and schedule to adapt to our needs. However not everyone knows how to program them, people must be prepared to understand how to make it correctly.

The purpose of this project is to create an easy and intuitive language in order to control the Surveyor SRV-1. This language has simple commands which control the movements of the robot and interact with the user or the environment. The programs will be written in XML files because there are a lot of resources to process these languages.

We have also developed a simulator to observe the behavior of the programs without the real robot.

In both cases, simulated or real robot, we can write programs editing XML files or through an interface that requires minimal computer knowledge.

In this document we are going to describe the specification of the language and different technologies integrated in this project such as the stereoscopic vision or voice communication with the robot.

Keywords: mobile robot, command language, 3D simulator, stereoscopic vision, robot programming.

Índice

1. Introducción y objetivos

Los robots son cada vez más autónomos y con funciones más generalizadas. Se ha evolucionado desde robots que se dedicaban a tareas específicas a robots que pueden realizar distintas tareas según se defina su comportamiento. Así, un mismo robot se puede emplear para tareas tan distantes como explorar un territorio inhóspito o transportar materias de un lugar a otro.

En un principio, uno de los objetivos principales de este proyecto era conseguir un lenguaje de comandos fácil e intuitivo con el que dotar de distintos tipos de funcionalidades al robot Surveyor SV-1. Este lenguaje debe ser fácil de programar e intuitivo para poder ser utilizado por personas que no tengan una preparación específica en programación de robots.

Sin embargo, cualquier usuario de este lenguaje deseará efectuar pruebas de los programas que haya escrito sin necesidad de utilizar el robot. Por ello, otro de los propósitos que marcamos para este proyecto fue la creación de un simulador, que permitirá comprobar el comportamiento del robot en distintas situaciones de superficie o distintos entornos sin necesidad de disponer del robot real. En el simulador se permite cambiar distintos parámetros del entorno como la fricción del suelo, la dureza de los materiales, etc., lo que nos facilita saber cómo se comportaría un robot en un terreno inhóspito, como podría ser un planeta sin explorar. Así, cuando el robot ejecute la tarea real sufrirá el mínimo daño posible o incluso se pueda descartar la tarea debido a que el riesgo sea demasiado elevado.

El proyecto, además de dar acceso al simulador, tiene la función de comunicarse con el robot real utilizando nuestro lenguaje.

Tanto en el uso del simulador como en del robot real, tendremos dos opciones de comunicación con el robot:

- La ejecución de una sola orden que realizará inmediatamente.
- La ejecución de una lista de órdenes que se irán ejecutando secuencialmente.

Además la comunicación mediante WIFI del Surveyor SV-1 nos permite mostrar en todo momento en pantalla que lo que está capturando el robot por sus cámaras.

Aprovechando que el robot usado, un Surveyor SRV-1, posee el módulo de visión estéreo, se decidió añadir otra funcionalidad, el análisis de las imágenes para que el robot pueda explorar el entorno y tomar decisiones. La visión estereoscópica nos permite calcular la distancia existente entre el robot y un punto del entorno utilizando la disparidad entre las dos imágenes, obteniendo así la tercera dimensión de las imágenes. Cuanto mejor se analice una imagen y más detalles podamos obtener de ella, mejor reproduciremos virtualmente el entorno que rodea al robot. Así, un robot que pueda reproducir su entorno, podrá simular que ocurrirá ejecutando determinados movimientos, y si estos movimientos son válidos o no para poder llevarlos a cabo o buscar alternativas.

Al avanzar el proyecto se fueron integrando algunas funcionalidades más que consideramos

útiles, como el poder interactuar con el robot a través de un micrófono. Así, el robot puede preguntar en determinado momento qué hacer y ser guiado por una persona mediante la voz y no solo ejecutando las órdenes mediante la interfaz.

La integración de las tecnologías necesarias para estas funciones se realiza mediante el uso de diferentes librerías. Así, para el desarrollo del simulador hemos utilizado el entorno TrueVision que nos permite simular entornos en 3D con modelos a escala. Para el control del robot real se ha empleado la librería AForge y para integrar el micrófono la librería SAPI. Todo el proyecto ha sido desarrollado en C# y Visual Studio 2008.

2. Arquitectura del sistema

La arquitectura de nuestra aplicación está formada por distintos módulos relacionados entre sí. En este apartado explicaremos la relación entre los módulos y su funcionamiento.

Distinguimos cinco módulos que engloban el funcionamiento de la aplicación:

- Lenguaje
- Simulador
- Visión
- Detección de voz
- Robot Surveyor SRV-1

El módulo de lenguaje es el encargado de realizar todas las tareas relacionadas con el lenguaje de comandos que utilizaremos. Realiza funciones de traducción para convertir cada orden de nuestro lenguaje al lenguaje de comandos del robot. Este módulo se relaciona con los módulos de visión y detección de voz porque ambos serán utilizados en ciertas órdenes del lenguaje que precisen estos módulos para decidir qué tipo de acciones realizar. Por ejemplo, una instrucción que interactúe con el entorno dando la orden al robot de buscar un objeto de un color necesitará al módulo de visión para utilizar las imágenes y buscar al objeto.

Por otra parte, también tendremos una relación entre este módulo y los módulos Surveyor y simulador debido a que estos ejecutarán las instrucciones en el robot que este módulo transforma al lenguaje objeto en caso del módulo Surveyor y a las funciones que realizan los movimientos en el simulador.

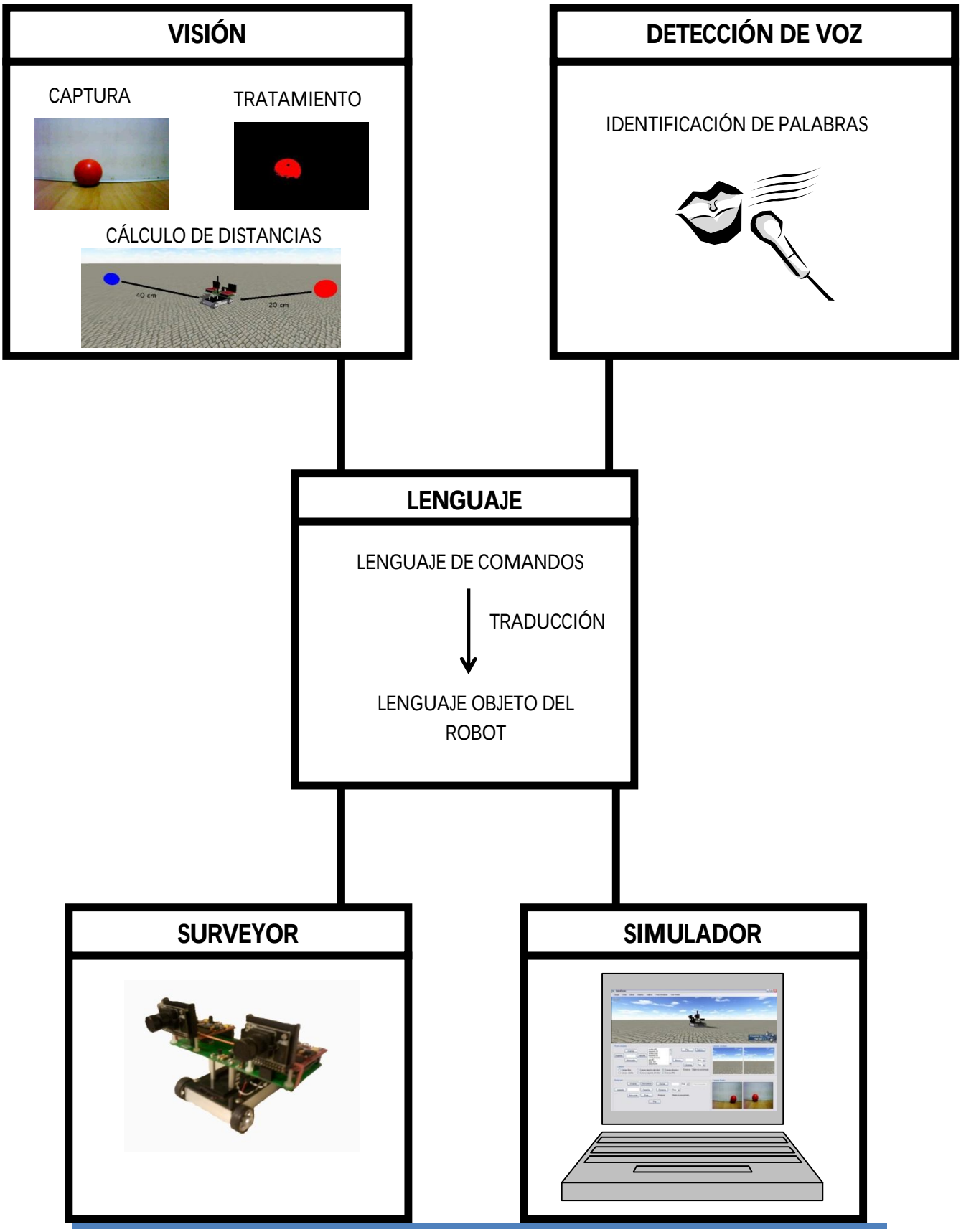
El módulo de visión contiene funcionalidades para realizar la captura y tratamiento de imágenes. Es el encargado de identificar un objeto de un color en las imágenes que captura el robot y calcular la distancia a dicho objeto.

El módulo de detección de voz se encarga de detectar palabras por el micrófono e identificarlas para decidir qué movimiento realiza el robot en el momento en que se precise la interacción del usuario.

El módulo Surveyor es el encargado de realizar la comunicación mediante WiFi con nuestra aplicación. Controlará el robot según le mandemos las órdenes del lenguaje objeto.

El modulo simulador contiene todo lo necesario para representar en 3D el robot y su entorno. Contiene funcionalidades para mover el robot dentro del entorno simulado, modificar las cámaras que enfocan al robot y ejecutar una serie de instrucciones de nuestro lenguaje.

El siguiente esquema muestra los módulos y la relación entre ellos:



2.1. Lenguaje

Uno de los principales objetivos de este proyecto es crear un lenguaje de comandos que permita a un usuario, sin necesidad de tener un conocimiento avanzado sobre robótica o programación, controlar el robot desde la aplicación tanto para el caso del simulador como para el robot real.

El lenguaje creado consta de cuatro tipos de instrucciones:

- instrucciones de movimiento
- instrucciones de cámara
- instrucciones de interacción con el entorno
- instrucciones de interacción con el usuario

Las instrucciones del primer grupo son instrucciones que permiten mover nuestro robot una distancia determinada por el usuario o girar el robot una cantidad determinada de grados. El grupo de instrucciones para las cámaras se utiliza para controlar las cámaras del simulador de forma que podamos modificar la posición desde la que observamos al robot. Las instrucciones de interacción con el entorno son aquellas que, al ser ejecutadas, toman decisiones en función del entorno que rodea a nuestro robot y, por último, las instrucciones de interacción con el usuario son aquellas que requieren la participación del usuario para tomar una decisión sobre la siguiente acción que ejecutará el robot.

Las instrucciones de las que está formado el lenguaje son las siguientes:

Movimiento:

acelera, atrás, izquierda, derecha y para

Cámaras:

eleva, gira y avanza

Interacción con el entorno:

búsqueda

Interacción con el usuario:

decisión

La ejecución secuencial de varias instrucciones formará una ruta. Estas rutas permitirán que se pueda crear un programa con el lenguaje y guardarlo en un fichero, de forma que podamos ejecutarlo en el robot el número de veces que deseemos sin la necesidad de reescribirlo. Esta información se manejará en archivos XML que nos permiten guardar y editar fácilmente las instrucciones.

2.2. Simulador

Como ya hemos explicado, otro de los principales objetivos de este proyecto es la creación de un simulador del comportamiento de robot Surveyor SRV-1 con el módulo de visión estereoscópica. Este simulador, debe reflejar el mismo comportamiento que el robot real, ya sea en cuanto a movimientos o referente a la ejecución de las rutas programadas en nuestro lenguaje de comandos. Esta parte de nuestro proyecto nos permite efectuar pruebas de nuestros programas, rutas y movimientos con la misma confianza que si lo hiciéramos con un robot real, ahorrando costes de electricidad, transporte y evitando poner el robot real en peligro.

Para que las pruebas sean lo más exactas posibles se permite la creación de objetos en cualquier momento de la simulación. También se dispone, para comodidad del usuario, de una serie de cámaras, seleccionables durante toda la ejecución, con las que se pueden observar los movimientos del robot desde cualquier ángulo.

El modelo del robot está representado en el entorno 3D por una malla que contiene el chasis y cuatro que contienen cada rueda. Cada malla representará un objeto en el entorno. Así cada nuevo objeto que creemos está representado por una malla independiente.

Al añadir la física del simulador se utilizará una función ya predefinida en el motor Newton que creará un vehículo indicándole cuáles son sus ruedas y cuál es su chasis. De esta forma, el robot constará de un chasis y de cuatro ruedas. Dando la orden de mover cada rueda en un sentido u otro conseguiremos que el modelo del robot modifique su posición moviéndose o girando.

2.2.1. Creación del entorno en el simulador

El simulador permite crear un entorno con obstáculos de diferentes formas y colores para que el escenario simulado sea lo más parecido posible al entorno donde se moverá el robot real o, si lo deseamos, para crear un escenario irreal donde experimentar situaciones que no se puedan llevar a cabo en un contexto real. Estas últimas podría servir para llevar a cabo los primeros ensayos, simplificando el escenario, o para probar los límites de validez de una ruta, complicando las condiciones para comprobar en qué momento fallaría el robot.

Para la creación de objetos dentro del escenario simulado disponemos de varias opciones:

- Creación rápida mediante formulario: Se mostrará un formulario al usuario donde el usuario podrá indicar el tipo de objeto, el color deseado y la posición donde se colocara el objeto.

- Creación mediante archivo XML: El usuario podrá cargar un archivo XML con el que cargar varios objetos a la vez definidos usando un sencillo lenguaje de comandos.

El usuario también tiene la opción de seleccionar y borrar objetos previamente creados, de esta forma el usuario podrá editar el escenario con total libertad.

2.2.2. Cámaras

Desde el principio el tema de las cámaras fue un aspecto importante, ya que si deseábamos realizar un simulador con una API de diseño 3D, nos interesaba tener una buena definición de la escena, de manera que se pudiera observar en cualquier momento donde estaba el robot y los movimientos que realizaba.

Nuestra primera idea fue trabajar con una cámara estática. Algo suficiente al principio, pero a medida que trabajábamos y desarrollábamos el simulador, nos enfrentamos con un problema: si utilizábamos una cámara estática predefinida podía ocurrir que se perdiera la representación de lo que estaba ocurriendo, saliendo el robot de la escena cuando la cámara estaba demasiado cerca del punto de origen del robot o, al alejarla del punto de origen, mostrar la escena con un nivel de detalle demasiado bajo para ser útil. Cada vez que queríamos probar algo teníamos que situar la cámara en el lugar deseado para obtener la vista deseada. Pensando en el usuario final de la aplicación, esto es inviable pues no podemos hacerle predefinir una posición sin que tenga una idea de la arquitectura del software 3D interno, además de la molestia que esto supone.

Llegados a este punto se decidió establecer una cámara que pudiera ser movida por el usuario de la misma manera que se puede mover en un videojuego: los movimientos de cámara, los que en informática gráfica se conocen como pitch y yaw, se pueden realizar con el ratón, a la vez que los avances de la cámara hacia adelante, atrás, izquierda y derecha se realizan con las teclas W, S, A y D del teclado sobre los ejes *x*, *y* y *z*. Esto supuso un gran avance que nos dio una gran libertad a la hora de observar la escena, pero pronto nos dimos cuenta que tal avance no suponía sino una pérdida de manejabilidad. El objetivo no era otro que simular el comportamiento del robot, con lo cual no nos interesaba tanto que el usuario pudiera "navegar" por la escena, como que pudiera observar y contrastar lo ocurrido con la realidad. En definitiva, nos pareció que esta cámara, siendo útil, podía no ser la mejor, por ello buscamos otras alternativas, sin llegar a desechar esta última.

La solución en nuestro caso fue una mezcla de nuestras anteriores ideas. Es por ello que el simulador posee múltiples opciones de cámara, todas ellas seleccionables en el apartado de cámaras del formulario principal.

Cámaras satélite y libre

Las cámaras satélite y libre son las que comentamos en los párrafos anteriores, siendo la libre la que puede mover el usuario mediante teclado y la satélite la que obtiene toda la vista de la escena desde arriba.

Cámaras izquierda y derecha

Representan las cámaras reales del robot, ya que nuestro objetivo principal es hacer la simulación lo más real posible. Con esta opción se puede ver el ángulo de

visión del robot desde el simulador y la visión teórica que posee en cada momento. Mediante estas dos opciones, el robot hace las capturas del entorno simulado.

Cámara dinámica

Este es un tipo de cámara con movimientos predefinidos. A medida que el robot avanza, se producen unos giros y avances predeterminados para dar una mejor visión de lo que está ocurriendo en la escena. Con estos movimientos se añade cierta espectacularidad y mayor sensación de movimiento.

Con esta opción pretendíamos dos cosas: primero, que no se perdiera de vista al robot y tener la certeza de tener siempre una buen ángulo de la escena, y segundo, mantener una visión atractiva para el usuario. Antes de comenzar a describir su uso y funcionamiento, tenemos que destacar que la particularidad de esta cámara es que siempre toma como referencia el centro del robot a partir del cual realiza todas las acciones.

Al trabajar con una API para entornos 3D, la implementación de giros y avances del punto de vista, se reduce a la aplicación de distintas funciones de geometría espacial. La propiedad LookAt de la cámara de la escena apunta siempre al centro del robot, aunque este esté trasladándose. Cómo hemos dicho antes, los movimientos son predefinidos: Cuando se produce un avance de los motores, la cámara comienza a reproducir un giro sobre el eje y, alrededor del objeto, calculando siempre la nueva posición de avance sobre la que gira, la nueva posición de la circunferencia descrita mediante trigonometría y avanzando un radián por cada vuelta del bucle de renderizado. Cuando se produce un giro del robot sobre su eje, la cámara deja de girar, comenzando un desplazamiento sobre el eje y en la posición en que se quedó con su anterior movimiento. La elevación en este caso, nos sirve para dar un mejor punto de vista del giro del robot. Si la elevación llega a unos límites y el robot continúa girando, deja de ascender, produciéndose un descenso. Hay que resaltar que ambos movimientos se concatenan, es decir, el giro se produce sobre la posición de y, dejada por la elevación y el ascenso se produce sobre el valor de x, z dejado por el giro. Con esto conseguimos que cada simulación posea unos movimientos de cámara distintos, dinámicos, no siendo siempre los mismos.

Por último señalar que este tipo de cámara es una buena elección, por parte del usuario, ya que no se tiene que preocupar por manejar la cámara, o perder el robot de vista. Además ofrece siempre un primer plano detalle de lo que ocurre durante la simulación.

Cámara XML

La cámara XML es una cámara que permite al usuario modificar el punto de vista de la cámara que enfoca al robot. Estos cambios se realizan con instrucciones del lenguaje de comandos creado para controlar el robot. Así estas instrucciones se incluirán en las rutas de movimientos haciendo que la cámara modifique su posición mientras el robot

realiza los movimientos e instrucciones dadas.

La idea de esta opción es que el usuario pueda manejar la vista de la escena sin engorrosas órdenes de teclado, predefiniendo los movimientos de cámara por su cuenta y poniéndolos en funcionamiento a la vez que carga una ruta.

Puesto que los movimientos de cámara pueden acabar antes de que se produzca la terminación de la ruta, la propiedad LookAt de la cámara se fija en el centro del robot, calculándose, como en la cámara dinámica, en cada vuelta del bucle de renderizado. De esta manera se puede seguir con esta opción, si la cámara ha terminado de realizar sus movimientos o incluso si ni siquiera se han predefinido, quedando de manera estática siguiendo al robot, sin perder un buen ángulo de visión de la escena.

Los movimientos que puede llevar a cabo esta cámara son los tres siguientes:

- **Elevación**: Es el mismo movimiento que puede realizar la cámara dinámica. Consiste en elevar la posición de la cámara con respecto al eje vertical, inclinando también la cámara para no dejar de presentar en pantalla al robot en ningún momento. Esta elevación se permite solo hasta cierta altura, ya que más allá de un determinado valor se pierde bastante detalle de lo que ocurre.
- **Giro**: Produce un giro alrededor del vehículo. Siguiendo la trayectoria de la circunferencia imaginaria sobre el eje y, realiza el giro especificado por los grados indicados por el usuario, siendo el radio de dicha circunferencia el que une la posición actual de la cámara con el centro del robot.
- **Avance**: produce un movimiento de cámara acercándose o alejándose del robot. El movimiento se produce sobre la recta que une el centro del vehículo y la posición actual de la cámara. En este caso, también marcamos límites de cercanía y lejanía a la posición del robot, para que los detalles de la escena no se pierdan.

Puesto que el usuario puede desear predefinir los movimientos de la cámara durante solo un tramo de la ejecución de su ruta, hemos incluido un comando en el lenguaje que permite cambiar la vista entre la cámara XML y la cámara dinámica para completar la visión de la escena en cualquier momento.

Por último añadir que esta opción quizá sea la más ventajosa y la que más partido saca a nuestro simulador. En cambio, recomendamos su uso una vez que el usuario esté relacionado con la aplicación, ya que, al contrario que la dinámica, no es la más sencilla y manejable.

2.2.3. Movimiento libre en el simulador

El robot simulado puede controlarse de forma manual mediante la interfaz principal o cargando un programa previamente escrito en el lenguaje de comandos que hemos diseñado.

Para controlarlo de forma manual el usuario usará los botones de la interfaz, que

permiten avanzar, retroceder y girar lo que se desee. Además existe la opción de comenzar la búsqueda o calcular la distancia de un objeto de un determinado color mediante la interfaz principal sin necesidad de utilizar el lenguaje de comandos.

2.3. Visión

El robot Surveyor con el que hemos trabajado posee dos cámaras en su parte frontal que unido a la movilidad que posee gracias a sus motores, convierte la posibilidad del reconocimiento de imágenes y, en general a los algoritmos de visión, en una herramienta potente y casi imprescindible en nuestro proyecto.

Antes de pasar a explicar los algoritmos y las técnicas empleadas, merece la pena detenernos en explicar las cámaras que incorpora el robot, su relación hardware y más que nada la obtención de imágenes para su posterior tratamiento.

2.3.1. Cámaras del Surveyor

El robot viene equipado con dos cámaras a color en su parte frontal de resolución 1280×1024 (1,3 megapíxeles). Cada cámara posee su propio procesador, para una mayor rapidez y un mejor tratamiento de la imagen.

El envío y recepción de los datos se produce mediante tecnología Wi-Fi, lo que permite al usuario contemplar en tiempo real las imágenes recogidas por la cámara.

Otras versiones del robot poseen una única cámara, pero este modelo incorpora un módulo de visión que proporciona al robot una cámara adicional, estas cámaras que están separadas una cierta distancia, nos dan la posibilidad de utilizar algoritmos de visión en estéreo para el cálculo de distancias o el reconocimiento de formas.

2.3.2. Tratamiento de imágenes

El tratamiento es el análisis de fotografías o imágenes en general, para un determinado fin, en el campo de la visión por computador. En nuestro caso, permite a robot reconocer el entorno que le rodea.

Las imágenes capturadas por las cámaras, se representan mediante matrices de píxeles en el modelo RGB. El modelo RGB es un modelo de color basado en la síntesis aditiva que permite definir un color mediante la mezcla de los tres colores primarios (rojo, azul y verde). Así, un color vendrá determinado por la intensidad que se le da a cada componente de color con valores entre 0 y 255 donde el blanco se conseguirá con la máxima intensidad en cada una de los valores (255, 255,255) y el negro con la mínima intensidad (0, 0,0).

Con este modelo podemos representar y obtener el color de cada pixel de la imagen. El conjunto de todos los pixeles será el que forme la imagen mostrada. Para cada pixel guardaremos la información de cada uno de los tres colores, lo que llamaremos de aquí en adelante, componente de cada color.

Para poder reconocer los objetos del entorno nos valdremos de los pixeles, de tal manera que los objetos vendrán delimitados por agrupaciones de pixeles con el mismo color o mejor dicho, con una componente parecida, ya que el escenario puede estar plagado de luces y sombras.

2.3.3. Visión estereoscópica

Como hemos explicado antes, gracias a las dos cámaras y a la separación entre ellas, se puede obtener una visión estereoscópica. Antes de comenzar vamos a intentar alcanzar una definición de este concepto.

La visión estereoscópica es la facultad que tiene un ser vivo de integrar las dos imágenes que está viendo en una sola por medio del cerebro. Por lo tanto, si tenemos dos imágenes tomadas desde posiciones ligeramente diferentes y las mostramos por separado a cada ojo, el cerebro es capaz de percibir la profundidad analizando la disparidad entre estas imágenes.

Vamos a reproducir este comportamiento gracias a la captura de dos fotogramas mediante ambas cámaras y así obtener la distancia a un objeto.

Una vez tratadas las imágenes, reconocido el objeto y su posición en ambas, tenemos los datos suficientes para analizar la diferencia entre posiciones y calcular la distancia. Todos los detalles de implementación se explican en posteriores apartados.

2.4. Voz

Para dar más funcionalidad al robot y al simulador se ha incluido en el proyecto una orden con la que el usuario puede interactuar con el robot en la toma de alguna decisión. Esta orden, denominada “Decisión”, permite, al ser ejecutada, que el usuario indique mediante un micrófono que debe de hacer el robot. Para definir el comportamiento que se llevará a cabo al ejecutar esta orden, necesitamos determinar qué palabras se pueden reconocer y la acción asociada a dicha palabra. Esto proceso se realiza mediante archivos XML. La herramienta que usaremos para el reconocimiento de las palabras será SAPI (Speech Application Programming Interface), una API desarrollada por la compañía Microsoft para permitir el reconocimiento de voz en aplicaciones de Windows. Esta API nos permite definir qué acciones realizar cuando una palabra es reconocida. Al detectar una palabra definida en el archivo XML mediante el micrófono se ejecutará un evento asociado que comprobará que palabra de las definidas se ha pronunciado y ejecutará su acción definida.

3. Hardware utilizado

Este proyecto se basa en dos elementos hardware para su funcionamiento completo, un computador y un robot móvil con visión estereoscópica. Sin embargo, uno de los objetivos de este proyecto es poder ejecutar pruebas y experimentos sin necesidad de usar dicho robot, por lo que el proyecto también permite ejecutarse, mediante un computador corriente, solo la parte de simulación.

Dicho computador, para que el proyecto funcione correctamente y de forma fluida, debe disponer de:

- Procesador Core2 Duo 2 GHz o Amd Athlon.
- Tarjeta gráfica dedicada de 128 MB
- Tarjeta de conexión inalámbrica 802.11g
- Sistema Operativo Windows XP o superior.
- 1 Gbytes de memoria RAM
- .NET FRAMEWORK 2.0 o superior.
- DirectX 9.0
- Motor gráfico TrueVision 6.5
- Librería AForge
- Microsoft SAPI (Speech Application Programming Interface)

En cuanto al robot, nosotros contamos con un robot Surveyor con un módulo de visión estéreo. Este robot dispone, básicamente:

- Dos motores eléctricos: Cada uno de los motores mueve las ruedas de uno de los lados del robot, ya que, en origen, el robot se movía mediante ruedas tipo oruga. Estos motores le permiten ir hasta un máximo de 40m/s
- Un módulo de comunicación WLAN 802.11g
- Dos módulos con cámara *SRV-1 BlackFin*, separados entre sí por 10.75cm. Cada uno de estos módulos está formado por:
    - Un procesador Blackfin BF537 a 500 MHz con 32 MB de memoria SDRAM y 4 MB de memoria flash.
    - Una cámara Omnivision OV9655 a 1.3 mega píxeles con una resolución máxima de 1280x1024 píxeles. Aunque puede capturar imágenes también a 640x480, 320x240 o 160x120 píxeles
- Batería recargable por enchufe.

Las dimensiones de Surveyor son:

- 120 mm de longitud x 100 mm de ancho x 80 mm de alto.
- El peso es de 350 gramos.

Además, cabe reseñar que el Surveyor SRV-1 puede ser controlado vía red inalámbrica

(WiFi 802.11b/g) con un alcance de hasta 100 metros en espacios cerrados. Alcance que se extiende hasta 1000 metros en espacios abiertos. Esta característica es una de las más importantes con respecto al uso en este proyecto

4. Diseño Software

En este apartado explicaremos detalladamente la implementación de los puntos clave de nuestra aplicación como la forma de implementar rutas, la ejecución de instrucciones del lenguaje de comandos o el tratamiento de imágenes.

También hablaremos del uso de las APIs utilizadas en la aplicación, como la API para implementar el simulador 3D o la API de reconocimiento de voz.

A continuación, explicaremos detalladamente las interfaces de las que consta nuestra aplicación para que cualquier usuario entienda perfectamente cómo utilizar la aplicación.

Por último, tendremos una organización de todas las clases usadas en la aplicación, especificando su función.

4.1. Lenguaje

4.1.1. XML/XML Schema

Al empezar a crear nuestro lenguaje de comandos barajamos distintas opciones para definir este lenguaje, como por ejemplo crear un analizador léxico y un sintáctico para que dada una gramática reconozca nuestro lenguaje. Otra opción consistía en usar archivos XML verificados con esquemas XML. Esta última opción nos permite reconocer el lenguaje muy fácilmente y además es un medio para transportar datos de un sistema a otro muy útil.

XML, eXtensible Markup Language, es un metalenguaje extensible de etiquetas que permite definir la gramática de lenguajes específicos como nuestro lenguaje de comandos para el robot con mucha comodidad.

XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML. Con un pequeño documento que especificara la estructura de nuestro lenguaje podremos comprobar un archivo XML para verificar que su estructura se ajusta a la de nuestro lenguaje.

Con estas dos herramientas podemos definir la estructura de nuestro lenguaje en un XML Schema y escribir programas con nuestro lenguaje en un documento XML. La estructura del archivo XML es la que define nuestra gramática y podríamos comprobarla leyendo el fichero XML y comprobando que en cada momento la orden que encontramos es la esperada, pero esta verificación de la estructura se realiza utilizando XML Schema de tal forma que no podemos comprobar que un archivo XML es correcto antes de que nuestro programa comience a leerlo. Si la verificación es incorrecta el archivo no se ajustará a nuestro lenguaje y no será una ruta valida.

4.1.2. Instrucciones

A continuación se describirán todas las instrucciones del lenguaje que hemos definido para controlar el robot. Estas instrucciones serán las mismas tanto para el simulador como para el robot real excepto las instrucciones que controlan el lenguaje

de cámaras que solo se usaran en el simulador por lo que el robot real las ignorará.

Para facilitar la explicación de las instrucciones u órdenes las hemos agrupado en cuatro grupos según su uso: movimiento, cámaras, interacción entorno e interacción usuario

Para cada grupo de ordenes se mostrará un cuadro resumen con su nombre, los parámetros de entrada que utiliza y una breve descripción de su función. A continuación, se detallará instrucción por instrucción cuales son los valores admitidos por sus entradas, una descripción más extensa y ejemplos de su uso.

4.1.2.1. Movimiento

Las instrucciones de movimiento permiten realizar el movimiento básico del robot tanto en el robot real como en el simulador.

Nombre	Entrada	Descripción
acelera *int*	distancia	Comando para mover hacia delante el robot.
para *int*	tiempo	Comando para parar el robot.
atrás *int*	distancia	Comando para mover hacia atrás el robot.
izquierda *int*	grados	Comando para realizar la rotación hacia la izquierda del robot.
derecha *int*	grados	Comando para realizar la rotación hacia la derecha del robot.

acelera *int* , **atrás** *int*

Entrada	***Descripción de la entrada***	***Intervalo de valores***
Distancia	Distancia en centímetros que recorrerá el robot.	Cualquier número positivo.

Comandos para mover una determinada distancia el robot hacia delante o hacia atrás. La distancia deberá estar siempre especificada. Si el valor de la distancia es 0 el robot se moverá indefinidamente.

Ejemplo:

```
// Moverá 40 centímetros el robot hacia delante.

acelera 40

// Moverá indefinidamente el robot hacia atrás.

atrás 0
```

para *int*

Entrada	***Descripción de la entrada***	***Intervalo de valores***
tiempo	Tiempo que el robot permanecerá parado en centésimas de segundo.	Cualquier número positivo.

Comando para parar un determinado tiempo el robot. El tiempo deberá estar siempre especificado. Si el valor de la distancia es 0 el robot no se parará.

Ejemplo:

// Para el robot durante 40.

para 40

izquierda *int* , **derecha** *int*

Entrada	***Descripción de la entrada***	***Intervalo de valores***
grados	Número de grados que rotará el robot.	1 -180

Comandos para rotar el robot un determinado número de grados en un sentido u otro. El número de grados deberá estar siempre especificado.

Ejemplo:

// Rotará el robot 40 grados hacia la derecha.

derecha 40

// Rotará el robot 90 grados hacia la izquierda.

izquierda 90

4.1.2.2. Cámaras

Los comandos para controlar las cámaras del simulador permitirán realizar distintos movimientos con la cámara que nos permitirán ver el escenario del simulador desde distintos puntos. Los comandos para controlar las cámaras del simulador solo funcionan con el simulador y no con el robot real que ignorará estos comandos.

Estos comandos de movimiento no son secuenciales por lo que si tras un comando de cámara aparece otro que modifique el movimiento de la cámara, esté último será el que se ejecute.

Nombre	Entrada	Descripción
gira *int*	grados	Comando para girar la cámara.
eleva *int*	distancia	Comando para elevar la cámara.
avanza *int*	distancia	Comando mover hacia delante o hacia atrás el robot.
cambiaCamara	Ninguna.	Comando para cambiar de cámara.

gira *int*

Entrada	***Descripción de la entrada***	***Intervalo de valores***
distancia	Número de grados que girara la cámara.	0-360

Comando para girar la cámara un número determinado de grados manteniendo como centro de giro el punto donde se encuentra el robot. El número de grados deberá estar especificado siempre.

Ejemplo:

// Girará la cámara alrededor del robot 125 grados.

gira 125

eleva *int* , **avanza** *int*

Entrada	***Descripción de la entrada***	***Intervalo de valores***
distancia	Distancia en coordenadas que se elevará o avanzará la cámara.	Cualquier número entero.

Comandos para elevar o mover la cámara hacia el robot. Dependiendo del valor negativo o positivo de la distancia la cámara se elevará o bajará y avanzará hacia el robot o se alejara de él. El valor de la distancia siempre debe estar especificado.

Ejemplo:

// Elevará la cámara 30 coordenadas.

eleva 30

// .Moverá la cámara 90 coordenadas hacia el robot.

avanza 90

cambiaCamara

Comando para cambiar de la cámara dinámica a la cámara XML. El comando cambiará a la cámara que no se está utilizando en ese momento.

4.1.2.3. Interacción con el entorno

Comandos del robot para que interactúe con el entorno buscando algo en él.

Nombre	Entrada	Descripción
búsqueda *Color*	color, distancia	Comando para iniciar la búsqueda de objetos de un color.

búsqueda *String,* **distancia** *int*

Entrada	***Descripción de la entrada***	***Intervalo de valores***
color	Color que del objeto que buscamos.	rojo, azul, verde
Distancia	Distancia a la que se quedará el robot de un objeto al buscarlo.	Cualquier número positivo.

Comando que comienza la búsqueda de objetos del color especificado en la entrada y una vez lo encuentre, se acerca a él. El color siempre debe de estar especificado.

Ejemplo:

// Comenzará a buscar objetos rojos.

Búsqueda rojo

4.1.2.4. Interacción con el usuario

Comandos del robot para que interactúe con el usuario.

Nombre	Entrada	Descripción
decisión	Ninguna.	Para la ejecución a la espera de una orden del usuario por micrófono.

decisión

Comando que deja el robot a la espera de que el usuario indique mediante un micrófono que hacer.

4.2. Rutas

En este apartado se explicará cómo se implementa y se guarda la información sobre las rutas creadas con el lenguaje de comandos.

Las clases que implementan las rutas están contenidas en el espacio de nombres GestionXML.

4.2.1. Ordenes

La clase ContenidoOrden servirá para representar una orden en nuestro programa. Esta clase contendrá dos atributos, un atributo "orden" para representar cual es el tipo de comando que queremos ejecutar y otro atributo "duración" que almacenará el valor numérico del comando (tiempo, grados,...).

El atributo orden será del tipo enumerado TipoOrden, que definirá los tipos de órdenes que tendremos para ejecutar en el robot. Cada orden coincidirá con un comando de nuestro lenguaje y servirá para representarlo en nuestro programa.

```
public enum TipoOrden { acelera, para, izquierda, derecha, atrás, ninguno, gira,
eleva, avanza, búsqueda, cambiaCamara, decisión };
```

4.2.2. Rutas en el programa

La clase ListaOrdenes implementará los métodos necesarios para poder definir listas que contengan elementos de la clase ContenidoOrden. Estas listas se utilizarán para definir las rutas del robot al ser ejecutadas.

4.2.3. Almacenamiento de rutas: XML y XML Schema

Las rutas quedarán almacenadas en ficheros XML validados y verificados con esquemas XML en los cuales esta especificados nuestro lenguaje.

El archivo esquemaRutas.xsd contiene la definición de cómo debe ser un archivo XML que contenga una ruta con nuestro lenguaje de comandos. Este archivo contiene el nombre de todos los comandos y el tipo de los valores que aparecen a continuación de los comandos.

A continuación mostramos el contenido de este archivo debido a que expresa la gramática y sintaxis de nuestro lenguaje:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/EsquemaRuta"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://tempuri.org/EsquemaRuta.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="ruta">
      <xs:complexType>
      <xs:sequence>
         <xs:choice maxOccurs="unbounded">
            <xs:element name="acelera" type="xs:integer" />
            <xs:element name="para" type="xs:integer" />
            <xs:element name="atras" type="xs:integer" />
            <xs:element name="izquierda">
            <xs:simpleType>
               <xs:restriction base="xs:integer">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="180"/>
               </xs:restriction>
            </xs:simpleType>
            </xs:element>
            <xs:element name="derecha">
            <xs:simpleType>
               <xs:restriction base="xs:integer">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="180"/>
               </xs:restriction>
            </xs:simpleType>
            </xs:element>
            <xs:element name="eleva" type="xs:integer" />
            <xs:element name="gira" type="xs:integer" />
            <xs:element name="avanza" type="xs:integer" />
            <xs:element name="busqueda">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="rojo"/>
                  <xs:enumeration value="azul"/>
                  <xs:enumeration value="verde"/>
               </xs:restriction>
            </xs:simpleType>
            </xs:element>
            <xs:element name="cambiaCamara" />
            <xs:element name="decision" />
         </xs:choice>
      </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Además del tipo, algunos valores tienen restricciones. Por ejemplo, el tipo que acompañara al comando “búsqueda” será un string, pero solo admitiremos tres valores: rojo, azul y verde.

Al ejecutar una ruta, primero se comprobará que se valida y verifica de acuerdo a nuestro lenguaje. A continuación mostraremos dos ejemplos, un archivo con una ruta valida y otro con una ruta no valida.

Ejemplo de ruta valida:

```
<?xml version="1.0"?>
<ruta xmlns="http://tempuri.org/EsquemaRuta.xsd">
   <atras>20</atras>
   <para>1</para>
   <izquierda>20</izquierda>
</ruta>
```

Esta ruta contendrá tres comandos y es válida porque cumple el esquema XML definido anteriormente.

Ejemplo de ruta no valida:

```
<?xml version="1.0"?>
<ruta xmlns="http://tempuri.org/EsquemaRuta.xsd">
   <atras>20</atras>
   <para>1</para>
   <izquierda>220</izquierda>
</ruta>
```

La restricción de que los comandos, izquierda y derecha, tengan valores de 0 a 180 no se cumple, por lo que esta ruta no es válida y no se ejecutara.

4.3. Simulador

Este apartado explicara cómo se ha llevado a cabo la implementación del simulador, la utilización de la creación de objetos e indicará como utilizar los comandos de cambio de cámara.

4.3.1. Creación del entorno en el simulador

Como ya se ha indicado, existen dos formas de crear objetos en el entorno simulado:

- Creación rápida mediante formulario: Para acceder a este formulario se debe pulsar la opción *Creación rápida* del menú de *Objetos*.

- Creación mediante archivo XML: Para cargar un fichero XML de creación de objetos debemos seleccionar la opción Objetos del menú Cargar, por lo que este archivo debe haber sido creado previamente. La forma de crear este tipo de archivos se explica en detalle más adelante.

En el menú *Objetos* también se dispone de las opciones seleccionar y borrar objetos previamente creados con las opciones *Seleccionar objeto* y *Borrar*.

- La primera acción selecciona un objeto de la lista de objetos creados, cambiando el color de la pieza a amarillo para permitir al usuario distinguirla de las demás.
- La opción *Borrar* nos permite, una vez que hayamos seleccionado un objeto, eliminarlo, con lo que desaparecerá del escenario de simulación y de la lista de objetos seleccionables.

Creación mediante archivo XML

En este apartado se explicará cómo se implementa y se guarda la información sobre los objetos creados con el lenguaje de comandos.

Las clases que implementan esta forma de creación de objetos están contenidas en el espacio de nombres GestionXML.

Representación de los objetos

Para crear lo objetos dentro de la simulación, utilizamos la creación de formas que incluye la librería. Una vez creado el objeto básico lo colocamos en las coordenadas que deseamos y le asignamos una textura para definir el color. Es por ello que nos centraremos en la definición dentro del archivo XML de esos objetos que luego crearemos al leer el fichero.

Para representar cada objeto a crear utilizaremos la clase ContenidoObjetos. Esta clase está formada por seis atributos:

- *Tipo*: Nos indica que forma tendrá el objeto, puede tomar los valores contenidos en el enumerado TipoObjeto.
- *Radio*: Es el atributo que determina el tamaño de la forma que queremos crear. Para una esfera indica el radio y para un cubo la longitud del lado.
- *coorX*, *coorY* y *coorZ*: Son las coordenadas del centro de la figura.
- *Color*: Este atributo nos indica de qué color será el objeto representado. Puede tomar los valores “rojo”, “azul”, “verde” y “amarillo”

Los objetos leídos desde el archivo se almacenan en una lista que, una vez finalizada la lectura del archivo, se recorrerá creando los objetos con los parámetros indicados en cada elemento. La clase ListaObjetos implementará los métodos necesarios para poder definir esta lista.

Almacenamiento de objetos: XML y XML Schemas:

Para crear un fichero XML con el que generar objetos, hay que seguir la gramática y la sintaxis adecuada, esto queda expresado en el archivo esquemaObjetos.xsd que mostramos a continuación.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="objeto">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="tipo" type="xsd:string" />
        <xsd:element name="radio" type="xsd:unsignedByte" />
        <xsd:element name="coorX" type="xsd:unsignedByte" />
        <xsd:element name="coorY" type="xsd:unsignedByte" />
        <xsd:element name="coorZ" type="xsd:unsignedByte" />
        <xsd:element name="color" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="listaObj" >
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="objeto" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Al cargar un archivo de creación de objetos, primero se comprobará que sea válido de acuerdo a nuestro esquema. A continuación mostraremos dos ejemplos, un archivo valido.

Ejemplo de objeto valido:

```
<ListaObj xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <objeto>
      <tipo>cubo</tipo>
      <radio>5</radio>
      <coorX>10</coorX>
      <coorY>35</coorY>
      <coorZ>5</coorZ>
      <color>azul</color>
  </objeto>
   <objeto>
      <tipo>esfera</tipo>
      <radio>9</radio>
      <coorX>10</coorX>
      <coorY>0</coorY>
      <coorZ>50</coorZ>
      <color>rojo</color>
  </objeto>
</ListaObj>
```

Este ejemplo creará un objeto con forma de cubo, de lado 5, en las coordenadas (10, 35, 5) de color azul y un objeto con forma de esfera, con radio 9, en las coordenadas (10,0,50) de color rojo. Como cumple el esquema XML definido anteriormente, este ejemplo es válido.

4.3.2. Cámara XML: Un lenguaje de cámaras

El tema de las cámaras ya ha quedado explicado en puntos anteriores, sin embargo, en este apartado indicaremos el funcionamiento de los comandos de la cámara XML, que, por ser la más compleja de uso, requiere una mayor explicación.

Como se ha dicho, esta cámara dispone de cuatro comandos:

- Elevación: Este comando XML recibe un parámetro *distancia* que indica la altura que se elevará la cámara. Ejemplo de uso: *<eleva> distancia </eleva>*
- Giro: Se indica en el parámetro *ángulo* los grados que se desea girar alrededor del robot. Ejemplo de uso: *<gira> ángulo </gira>*
- Avance: El comando recibe un valor como entrada, la distancia que se desea mover la cámara. Dicho valor es el valor del cateto de avance de la coordenada x, mediante el cual calculamos la coordenada z, por el teorema de Pitágoras obtenemos hipotenusa que será el incremento sobre la trayectoria de avance. Ejemplo de uso: *<avanza> distancia </avanza>*
- Cambio cámara dinámica/cámara XML: Se usa para cambiar de una cámara a otra en cualquier momento. Ejemplo de uso: *<cambiaCamara/>*

4.3.3. Simulaciones físicas: Motor Newton

Para representar las fuerzas físicas dentro del simulador utilizamos el motor Newton incluido dentro del motor gráfico TrueVision. Este motor nos permite definir dentro de la simulación todos los parámetros físicos de los que dependen los movimientos del robot, tales como la fricción, el peso, la fuerza de la gravedad, etc..

4.3.4. La clase simulador

El simulador y su entorno se generan desde la clase Simulador, cuyos métodos podemos clasificar en varias secciones según su funcionalidad:

- Constructora

public Simulador(System.Windows.Forms.PictureBox canv, MainForm form)
Crea el objeto Simulador, para ello primero crea todo lo necesario para el entorno 3D y las cámaras para después llamar a los métodos de inicialización respectivos.

- Inicialización y finalización

InitPhysicsMaterials()	Inicializa los parámetros físicos de los materiales que se van a usar en el escenario. Definiendo aquellos que van a afectar a la interacción de cada material con el resto del entorno
InitMaterials()	Inicializa las características de los materiales que no afectan a su interacción con el resto de materiales, tales como la reflexión de la luz.
InitTextures()	Carga las texturas desde los ficheros para poder ser usadas en cualquier momento por los objetos del simulador.
InitEnvironment()	Inicializa el entorno del escenario, es decir, las texturas del cielo, y la forma del mundo simulado
InitPhysics()	Inicializa las físicas para el motor Newton. Define el modelo de fricción y la fuerza de la gravedad
InitLandscape()	Crea el terreno por donde se moverá el robot, definiendo su tamaño, textura, material y su orografía.
InitObjects()	Se inicializan los objetos del escenario, en nuestro caso se llama a la creación del robot.
InitLights()	Inicializa las luces en el escenario. En nuestro caso genera un foco de luz que representa el sol
cerrarSimulador()	Cambia el valor de las variables necesarias para que el simulador deje de renderizar y se pueda cerrar de forma segura

- Bucle de renderizado

GameLoop()	Contiene el bucle encargado del renderizado de todos los objetos del simulador y de mantener funcionando todo el simulado.

- Control de entrada

CheckInput()	Comprueba las acciones de teclado o del ratón para controlar el movimiento de la cámara libre
fijarCamara()	Controla el booleano para activar o desactivar la cámara libre

- Control de la cámara XML

giraXmlCam(int g)	Gira la cámara XML un ángulo de *g* grados
elevaXmlCam(int h)	Eleva la posición de la cámara XML incrementando su altura en *h*
avanzaXmlCam(int v)	Avanza o retrocede la posición de la cámara XML incrementando o decrementando la distancia con respecto al robot según el valor de *v*
cambiaCamara()	Cambia el tipo de cámara activo

- Control del robot

acelera(float vel)	Activa los motores simulados del robot, ajustándolos a la velocidad indicada por el valor de *vel*
para()	Para los motores y activa los frenos del robot
gira(int fase)	Hace girar el robot, para ello activa los motores del robot, haciendo girar las ruedas de un lado hacia adelante y las del otro hacia atrás. El valor de *fase* indica si el robot debe girar hacia la izquierda (*fase* = 1) o a la derecha (*fase* = -1)

- Creación del robot

creaRobot()	Crea las simulaciones de las piezas del robot (definiendo sus texturas, sus parámetros físicos y sus posiciones), las une, configura el centro de masa y las fricciones entre las piezas.

- Captura y tratamiento de imágenes

ParImagenes captura()	Es el método encargado de capturar las imágenes de las cámaras virtuales del robot, devolviéndolas usando la clase *ParImagenes*
TV_3DMATRIX transpose(TV_3DMATRIX atrasponer)	Transpone la matriz *atrasponer* devolviendo el resultado
float distancia(Bitmap imageL, Bitmap imageR, Label label3, String color)	Calcula la distancia que hay entre el robot y el objeto de color indicado en *color*, usando las imágenes *imageL* e *imageR,* devolviendo el resultado calculado.
TV_2DVECTOR posicionPuntoImagenUnsafe(Bitmap b, String color)	Devuelve las posiciones de los puntos del color *color* que se encuentren en la imagen *b*

- Creación, selección y eliminación de objetos

creaObjeto(TipoObjeto tipo, int radio, int x, int y, int z, String color)	Crea un objeto de tipo *tipo*, una esfera de radio *radio* o un cubo de lado *radio*, en las coordenadas *(x,y,z)* con el color *color*
selecObjeto(int p)	Selecciona y marca, cambiando la textura, el objeto que se encuentre en la posición *p* de la lista de objetos creados
TVMesh[] getListaObjetos()	Devuelve la lista de objetos creados en la simulación
TVMesh getObjeto(int i)	Devuelve el objeto que se encuentre en la posición *i* de la lista de objetos creados
elimElem()	Elimina el objeto seleccionado mediante el método *selecObjeto,* siempre que se haya hecho ya esa selección previa

4.4. Visión

En este apartado vamos a exponer de manera más extensa las ideas presentadas en apartado 2.4, haciendo más hincapié en cuanto a la implementación.

4.4.1. Cámaras del Surveyor

Gracias a la librería AForge, podemos obtener las imágenes recogidas por las cámaras, para su posterior tratamiento

En nuestra aplicación, la recepción de los datos enviados por las cámaras comienza cuando la aplicación se conecta a la red del robot. Una vez hecho esto, podemos pulsar el botón conectar, para establecer tráfico de datos con el robot. Desde el formulario principal puede verse que la conexión se ha realizado con éxito cuando en el apartado "cámaras reales", puede apreciarse la recepción de las imágenes en tiempo real de ambas cámaras.

4.4.2. Tratamiento de imágenes

Como se comentó en el apartado 2.4.3, necesitamos determinar los pixeles que corresponden a un objeto mediante su color. Para poder detectar si un pixel es de un determinado color usaremos unos determinados umbrales, que serán la diferencia de la componente del color que queremos detectar con respecto al resto de componentes. Dicha diferencia marcará el límite a partir del cual decidiremos si un pixel es de un color o no.

El motivo de este proceso es que en la realidad un objeto de color rojo no será totalmente puro, es decir, al obtener una imagen de ese objeto y observar uno de sus pixeles, el valor de las componentes no será de 255 para el rojo y de 0 para el azul y el verde, pero si tendrá un valor alto para la componente roja y un valor bajo para la componente azul y verde.

Algoritmo de tratamiento

Este algoritmo es básico, ya que solo discrimina pixeles en función de los umbrales para saber si son rojos, azules o verdes, por lo que el objeto del color que buscamos debe estar perfectamente determinado en la escena de la imagen. Esto hace que si algún pixel es del mismo color que el objeto que buscamos y está alejado de él, el algoritmo no calcule bien la distancia.

Por este motivo, existe un grave problema al tratar los fotogramas recibidos: el ruido producido por la calidad de la imagen. Así, en una imagen en la que tenemos un objeto rojo inequívocamente identificado, un pixel rojo causado por el ruido en otra parte de la imagen hará que el algoritmo calcule mal la posición del objeto.

El algoritmo empleado discrimina estos pixeles. Para ello, recorreremos la imagen pixel por pixel y, si el este es del color requerido, comprobaremos si sus vecinos

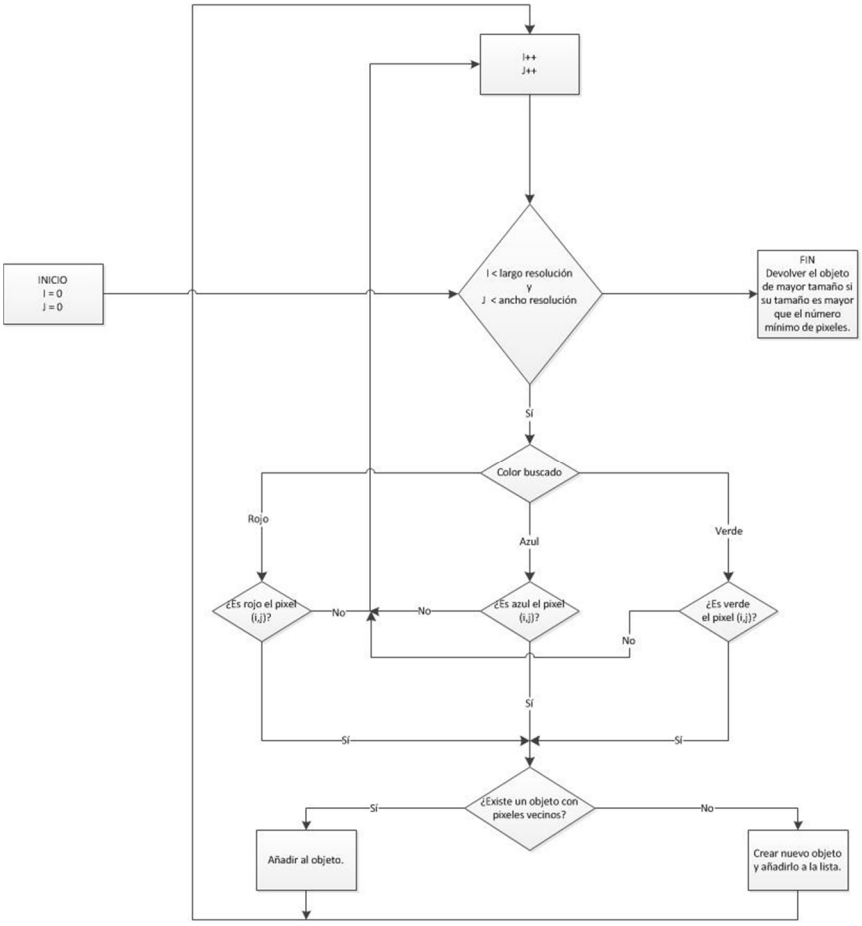
(aquellos que son adyacentes a él) son del mismo color. Si esto es así, añadiremos el pixel actual al objeto que forman los anteriores. Una vez concluya el algoritmo tendremos un lista de objetos de la cual nos quedaremos con el objeto mayor, discriminando totalmente todos los pixeles que forman parte del ruido.

Con esta otra versión del algoritmo, surge otro problema: el ruido hace que si en una escena en la que no hay ningún objeto del color buscado, un pixel de ruido tiene ese color, el robot lo identificará como un objeto, ya que es lo único que se ha reconocido que coincida con el color deseado.

Para evitar este efecto, se definirá un número de pixeles mínimo a partir de cual se considerara si un objeto es válido o no. Así, si el objeto mayor obtenido de la lista tiene un número menor de pixeles que el indicado, se discriminará también ese objeto y se indicará que no se han encontrado objetos del color que buscamos.

En el caso de detectar un objeto valido, sumaremos las posiciones de todos sus pixeles y dividiremos entre el número total de pixeles que forman ese objeto. De esta forma calcularemos la posición central de ese objeto, teniendo así una sola posición con la que poder comparar la otra imagen.

En la figura siguiente mostramos un diagrama de flujo del funcionamiento de nuestro algoritmo.



4.4.3. Calibración de los umbrales

La selección de los umbrales adecuados es un factor determinante a la hora de tratar las imágenes. Por ejemplo, queremos detectar pixeles de color rojo y suponemos los siguientes umbrales:

- Umbral rojo-azul = 150
- Umbral rojo-verde = 120

Con ellos, los pixeles que tengan una diferencia entre la componente roja y la componente azul de 150 y de 120 entre la componente roja y la verde, serán considerados de color rojo, por lo que serán pixeles que estamos buscando.

Nuestra aplicación posee una interfaz de calibrado, donde se pueden fijar las diferencias entre las distintas componentes de color para establecer los umbrales. Además se podrá seleccionar el número de pixeles mínimo que desea el usuario para reconocer los objetos. Una vez fijados estas constantes, se muestran sendos mapas de bits que reflejan los pixeles reconocidos por ambas cámaras para los valores establecidos anteriormente.

Esto nos permite poder ajustar mejor los valores para el tratamiento de la imagen antes de que el robot pase a ejecutar los algoritmos de visión.

Por otro lado, nos interesa disponer de un escenario de trabajo que se encuentre "limpio" de sombras, reflejos o incluso otros objetos que puedan interferir en el análisis y ser identificados erróneamente por los umbrales. Para ello, hemos implementado otro algoritmo de calibración mediante el cual el robot realiza un muestreo cada 5 segundos del entorno, dando a conocer al usuario la imagen reconocida para los umbrales seleccionados y con la posibilidad de cambiarlos dinámicamente.

Con ello podremos comprobar si el escenario es adecuado para obtener un buen funcionamiento del robot.

4.4.4. Visión estereoscópica

La visión estereoscópica para el cálculo de distancias mediante dos imágenes, es un mecanismo basado en triangulación. Para ello necesitamos encontrar el mismo objeto en ambos fotogramas. Para que no se produzcan errores, se necesitan reconocer objetos con colores claramente diferenciados del escenario. Como se comentó en anteriores apartados, tras realizar algunos ajustes relativos a los pixeles que se quieren reconocer mediante una interfaz de calibración, se procede al reconocimiento de los mismos mediante varios barridos de la imagen. Una vez encontrados los pixeles relativos al objeto, se dispone de la información necesaria para el cálculo de distancias. Esto es lo que se llama un análisis de la disparidad.

Siendo B la separación entre las cámaras, λ la distancia focal y Z la distancia total, si

la imagen de la primera cámara coincide con el sistema de coordenadas absoluto:

1. Primera Cámara: $X_1 = \frac{x_1}{\lambda}(\lambda - Z_1)$

2. Segunda Cámara: $X_2 = \frac{x_2}{\lambda}(\lambda - Z_2)$

Puesto que la distancia Z queremos que sea la misma, dejamos una sola incógnita, resolviendo el sistema por sustitución:

$$X_1 + B = \frac{x_2}{\lambda}(\lambda - Z)$$

Obteniendo, al resolver, la ecuación que nos proporciona la distancia:

$$Z = \lambda - \frac{\lambda B}{x_2 - x_1}$$

4.5. Voz

Para la implementación del reconocimiento de voz utilizamos SAPI. Esta herramienta utiliza un archivo XML para definir las palabras que se quieren reconocer y las asocia a las acciones deseadas.

En este apartado se explica cómo hemos definido las palabras que hemos utilizado, mediante el archivo XML, y las acciones que se llevan a cabo al utilizar la orden *Decision*.

Archivo XML:

Para poder reconocer palabras debemos definir reglas con las palabras que queremos sean reconocidas. En el archivo gramar.xml definimos las cuatro palabras que reconocerá nuestro robot: one, two, three y four. En la versión de SAPI usada no se admite el idioma español por lo que hemos escogido palabras en inglés fácilmente pronunciable y conocidas por todos.

De esta forma la definición de este archivo será la siguiente:

```
<GRAMMAR LANGID="409">

<DEFINE>
<ID NAME="RID_One" VAL="0"></ID>
<ID NAME="RID_Two" VAL="1"></ID>
<ID NAME="RID_Three" VAL="2"></ID>
<ID NAME="RID_Four" VAL="3"></ID>
</DEFINE>

<RULE NAME="rule1" ID="RID_One" TOPLEVEL="ACTIVE">
<P>one</P>
</RULE>

<RULE NAME="rule2" ID="RID_Two" TOPLEVEL="ACTIVE">
<P>two</P>
</RULE>

<RULE NAME="rule3" ID="RID_Three" TOPLEVEL="ACTIVE">
<P>three</P>
</RULE>

<RULE NAME="rule4" ID="RID_Four" TOPLEVEL="ACTIVE">
<P>four</P>
</RULE>

</GRAMMAR>
```

Orden "Decisión":

La orden decisión para la ejecución de una ruta, parando el timer en el caso del simulador y deteniendo el robot en el caso real, y se mantiene a la espera hasta que se detecta que se ha pronunciado una de las cuatro palabras definidas.

Cada palabra definida tiene una orden preestablecida. La siguiente tabla muestra que acción se ejecutará con cada palabra.

Palabra reconocida	Acción a realizar
one	Mueve el robot hacia delante 10 cm.
two	Mueve el robot hacia atrás 10 cm.
three	Gira el robot 90 grados hacia la izquierda.
four	Gira el robot 90 grados hacia la derecha.

El proceso al reconocer una palabra, tanto en el simulador como en el robot, será indicar que la orden actual es la correspondiente a la palabra reconocida y continuar con la ejecución de la ruta.

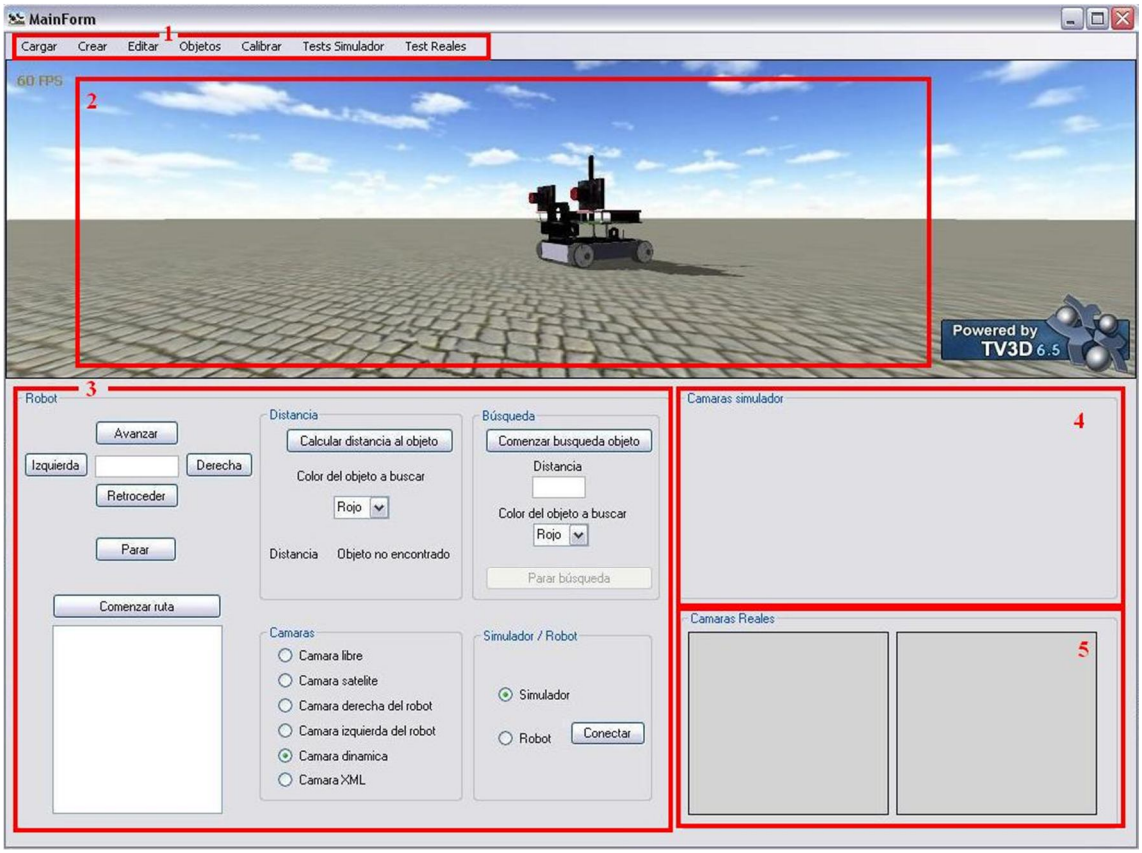
4.6. Interfaces

En este apartado explicaremos con imágenes y ejemplos la interfaz gráfica que proporciona nuestro programa para que el usuario pueda interaccionar con el robot y el simulador.

4.6.1. Interfaz principal

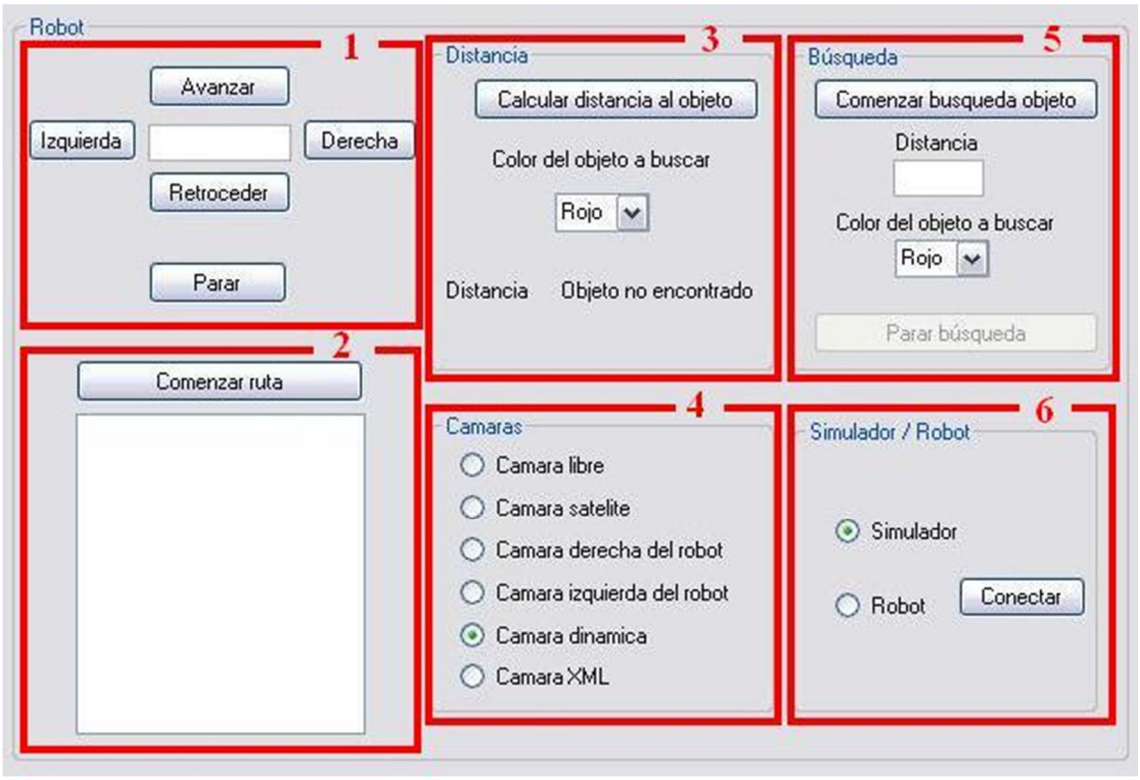
La interfaz principal contendrá los siguientes elementos:

1. El menú que contendrá el acceso a las demás interfaces.
2. El simulador del robot.
3. Un cuadro de control para el robot.
4. Dos imágenes que mostrarán lo que captura el robot en el simulador.
5. Dos imágenes que mostrarán lo que captura el robot en la realidad.



Cuadro de control:

Este cuadro contiene todos los botones necesarios para interactuar con nuestro robot simulado o el robot real. Explicaremos cada una de las zonas marcadas en la imagen a continuación:



1. **Control de los movimientos del robot**

Indicando en el campo de texto la distancia o los grados y pulsando el botón respectivo, moveremos el robot.

Pulsando el botón "Parar" el robot se parará y dejará de ejecutar el movimiento que estuviera realizando.

2. **Lista de órdenes**

Al cargar una ruta, está se mostrará en el campo de texto. Al pulsar sobre el botón "Comenzar ruta" se comenzará a ejecutar la ruta.

3. **Cálculo de la distancia**

Al pulsar el botón "Calcular distancia al objeto" se mostrará la distancia al objeto del color indicado en el desplegable.

4. Elección de las cámaras

Este cuadro contiene todos los tipos de cámaras que tenemos para la visualización del robot simulado. Pulsando sobre uno de ellos se cambiará inmediatamente a la cámara elegida.

5. Búsqueda

Pulsando el botón "Comenzar búsqueda objeto", el robot empezará a buscar el objeto del color especificado en el desplegable y se acercará la distancia indicada en el campo de texto a dicho objeto.

6. Simulador / Robot

En este cuadro podremos elegir entre controlar el robot real o el simulado. Todos los cuadros explicados anteriormente realizarán acciones en el simulador o en el robot real dependiendo de la casilla marcada en este cuadro.

Además, el botón "Conectar" conectará el robot real con la aplicación.

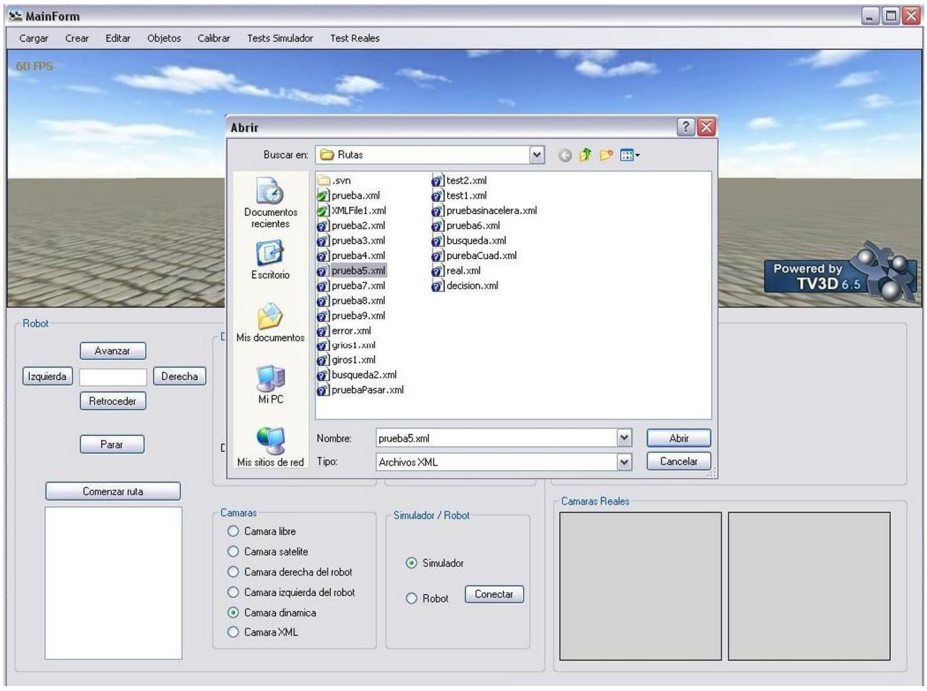
4.6.2. Interfaces para el uso de rutas y objetos

La información de las rutas se guarda en archivos XML por lo que nuestro programa ofrecerá interfaces para cargar, editar y crear estos archivos. Lo mismo ocurrirá con los archivos que contienen la información sobre los objetos del entorno.

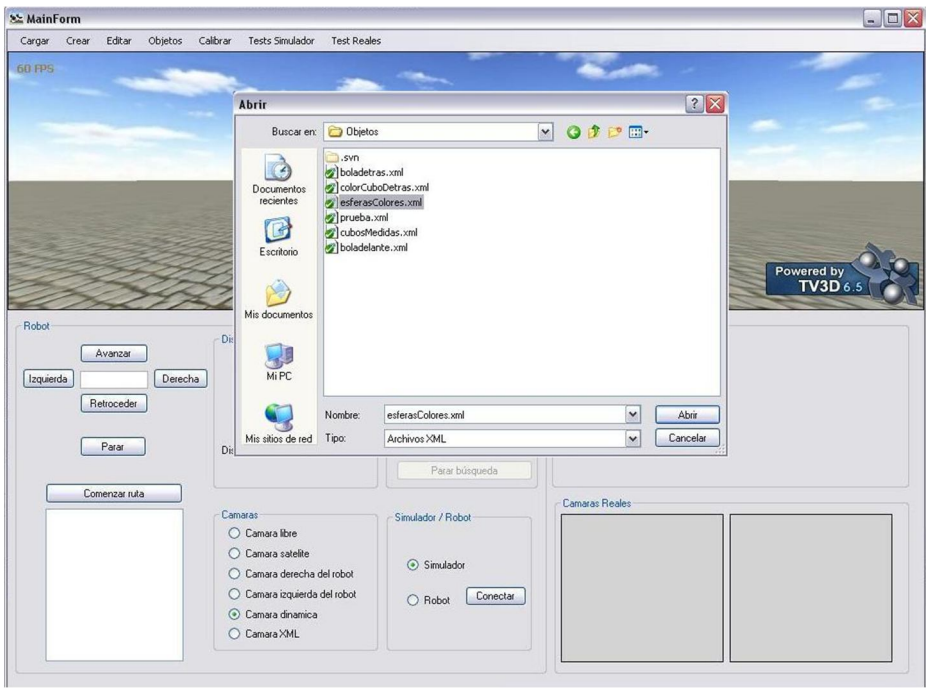
Además de permitir la edición de archivos XML, daremos soporte para poder crear tanto objetos como rutas sin editar estos archivos para usuarios que desconozcan esta tecnología.

Cargar archivos XML con rutas u objetos:

En el menú de la interfaz principal tendremos la opción Cargar > Rutas que nos mostrará una ventana donde podremos elegir el archivo XML que queremos cargar.

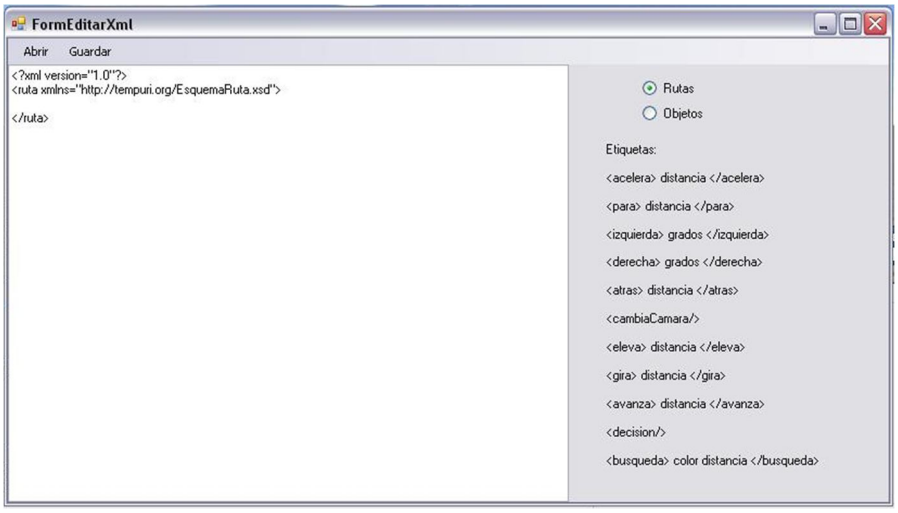


Ocurre lo mismo para los objetos con la opción Cargar > Objetos.



Editar archivos XML con rutas u objetos:

Pulsando en la opción *Editar* del menú principal llegaremos a la siguiente interfaz:



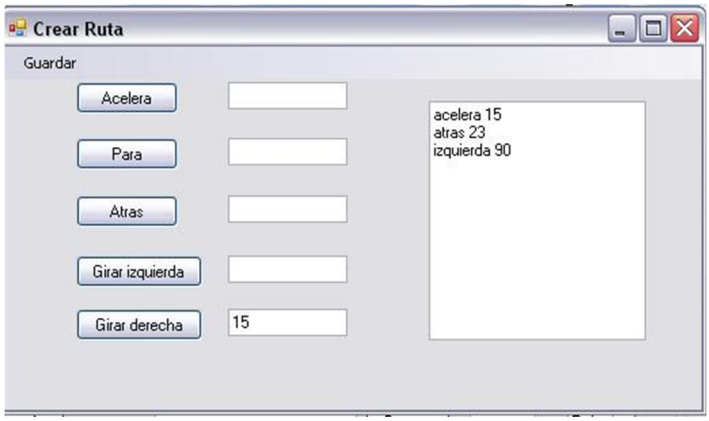
En ella podemos editar con código XML tanto las rutas como los objetos. Podemos crear un archivo nuevo o abrir uno ya existente y editarlo.

Pulsando en "Objetos" podremos crear y editar los archivos XML que contienen la información de los objetos.

Crear rutas sin editar archivos XML

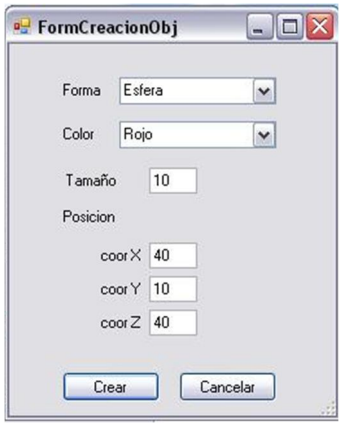
La interfaz de crear rutas está orientada a personas que no conozcan como editar archivos XML. En ella podremos crear una ruta poniendo el valor que queremos en una orden y pulsando el botón correspondiente. Repitiendo este proceso iremos creando una ruta que se guardará en un archivo XML pero sin haberlo editado directamente.

Accederemos a esta interfaz a través de la opción "Crear" del menú principal.



Crear objetos sin editar archivos XML:

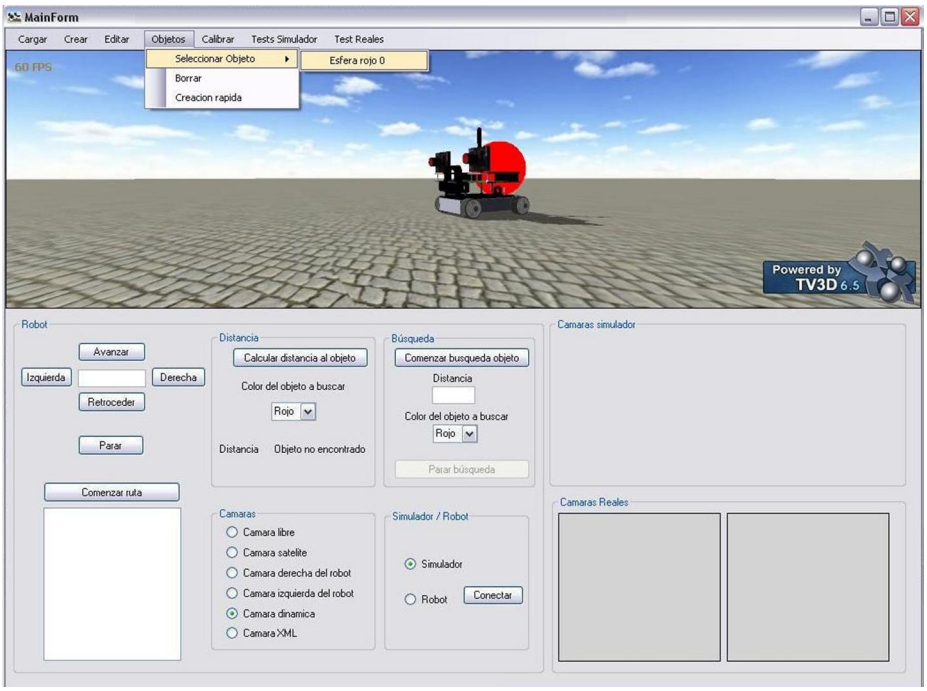
La opción Objetos > Creación rápida nos permitirá crear objetos sin editar archivos XML. Para ello rellenaremos el siguiente formulario con la forma del objeto, color, tamaño y coordenadas.



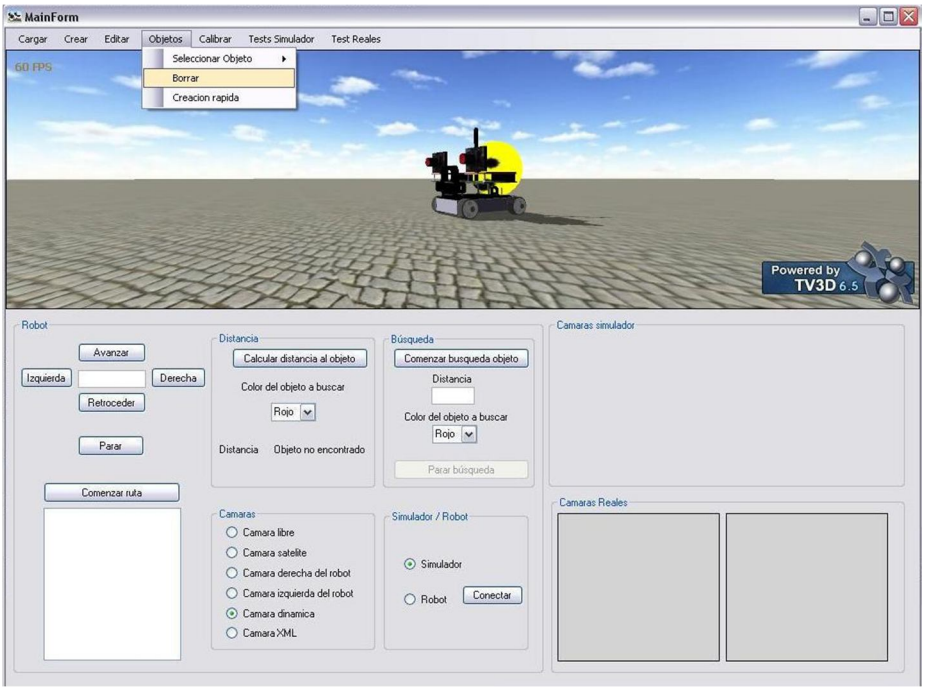
Seleccionar y borrar objetos:

En el menú Objetos del menú principal tendremos dos opciones más aparte de Creación Rápida. Estas opciones son Seleccionar Objeto y Borrar.

Seleccionar Objeto nos permite seleccionar un objeto de la lista que muestra todos los objetos creados en ese escenario.



Una vez seleccionado el objeto, con la opción Borrar lo eliminaremos del escenario actual.

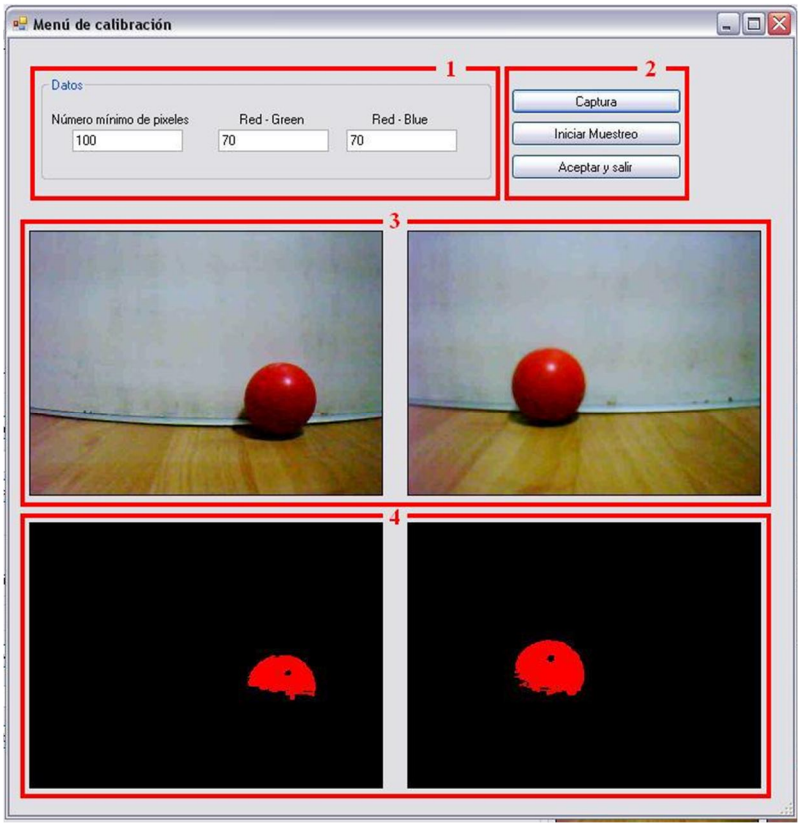


4.6.3. Interfaz de calibración

En el menú principal tenemos la opción Calibrar que abrirá la interfaz donde podremos calibrar los umbrales de color y el número mínimo de pixeles necesarios para considerar un objeto como tal. Para ello realizaremos capturas de las imágenes de las cámaras del robot en un instante dado y mostraremos un mapa de bits con los bits reconocidos como el color elegido según los umbrales y el número mínimo de pixeles.

Este procedimiento podemos hacerlo de dos formas, iniciando un muestreo que cada cinco segundos captura las imágenes del robot y muestra cuales serían los pixeles reconocidos o haciendo una única captura y mostrando dichos pixeles. La función del muestreo es examinar un escenario moviendo el robot manualmente y así poder asegurar que no existe ningún objeto además del que queremos buscar.

Para acceder a este formulario, pulsaremos sobre la opción Calibrar del menú principal, donde tendremos tres opciones, una para cada color. Si pulsamos en uno de los tres colores accederemos al formulario de calibración de dicho color.



Datos para el tratamiento de imágenes:

En estos campos de texto podremos cambiar los valores de los umbrales y el número mínimo de pixeles.

Capturas, muestreo y cambiar valores:

El botón Captura hará una única captura de las imágenes y las tratará para mostrar cómo se reconocerían los pixeles buscados.

El botón Iniciar Muestreo iniciará la captura y tratamiento de imágenes cada cinco segundos. En dichos intervalo podremos cambiar el valor de los umbrales y el número mínimo de pixeles.

Por último, el botón Aceptar y Salir saldrá del formulario y retorna al formulario principal cambiando los valores de los umbrales y el número mínimo de pixeles con los datos introducidos en el formulario de calibración para realizar la búsqueda de objetos

Imágenes reales:

Aquí se mostrarán las imágenes que está capturando el robot en tiempo real.

Imágenes tratadas:

Estas imágenes muestran el resultado de tratar las imágenes de una captura.

4.7. Organización general de las clases

En este apartado daremos una visión de cómo está organizada la implementación de nuestro proyecto, indicando espacios de nombre, clases y métodos principales. Para ello indicaremos el nombre de cada espacio de nombres y a continuación el nombre de cada clase y una breve explicación de cuál es su función.

4.7.1. Proyecto

Espacio de nombres que contendrá las clases principales de la aplicación.

MainForm

MainForm implementa el formulario principal de nuestra aplicación donde se muestran todos los controles del simulador y el robot. En esta clase también se implementa la ejecución de rutas (3.3). Se explica más detalladamente el funcionamiento de los controles en el apartado Interfaces (3.1.1).

Program

Esta clase se implementa automáticamente por la herramienta Visual Studio y sirve para comenzar la ejecución de la aplicación.

4.7.2. Formularios

Espacio de nombre que contendrá las clases que implementan los formularios utilizados en la aplicación.

FormCalibrar

Formulario para realizar la calibración del tratamiento de imágenes (3.1.3).

FormCreacionObj

Formulario para la creación de objetos en el escenario (3.1.2).

FormCrearRuta

Formulario para la creación de rutas (3.1.2).

FormEditarRutasXml

Formulario para la creación y edición de rutas y objetos (3.1.2).

FormImagenGrande

Formulario para ampliar las imágenes de la calibración (3.1.3)

4.7.3. Gestión XML

Espacio de nombres que contendrá las clases que gestionarán los archivos XML.

ContenidoObjeto

Clase que implementa la información que se necesita de un objeto.

ContenidoOrden

Clase que implementa la información que se necesita de una orden.

ExcepcionXML

Clase que implementa las excepciones producidas por la validación de archivos XML con XML Schema.

LectorEscritorXML

Clase que implementa los métodos necesarios para leer y escribir archivos XML.

ListaObjetos

Clase que implementa la estructura de la lista de objetos.

ListaOrdenes

Clase que implementa la estructura de la lista de órdenes.

ValidacionXML

Clase que implementa los métodos para validar archivos XML

4.7.4. Imágenes

Espacio de nombres que contendrá las clases relacionadas con el tratamiento de imágenes.

Coordenada:

Clase que implementa la información de una coordenada.

Imagen:

Clase que implementa los métodos para el tratamiento de imágenes.

ParImagenes:

Clase que implementa la información sobre una captura de imágenes.

4.7.5. Motor

Espacio de nombre que contendrá la clase para la implementación de nuestro simulador.

Simulador:

Clase que implementa los métodos necesarios para la implementación del simulador y los movimientos tanto del robot como de las cámaras dentro de este.

4.7.6. Robot real

Espacio de nombre que contendrá la clase para la implementación de la comunicación con el robot.

Robot:

Clase que implementa todos los métodos para comunicarse y mover el robot.

4.7.7. Sonido

Espacio de nombre que contendrá la clase para la implementación del control del micrófono.

Micro:

Clase que implementa todos los métodos para utilizar el micrófono y reconocer palabras.

5. Tests

En este apartado vamos a describir algunos de los Test que incluye el simulador para comenzar a familiarizarse con la aplicación.

En primer lugar, tenemos dos tipos de test distintos: los del robot real y los del simulador. Cada uno de ellos va a poner en práctica un comportamiento tanto en la simulación como en la realidad.

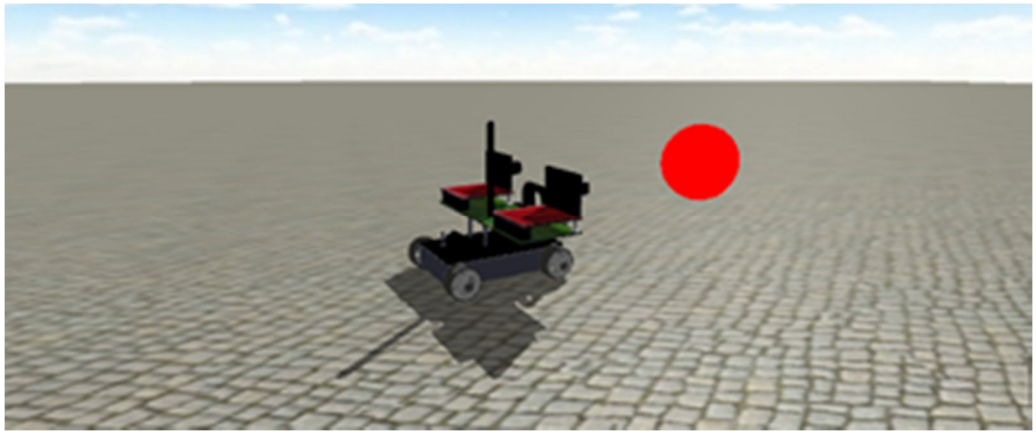
Todos estos test son accesibles desde de la interfaz del formulario principal.

5.1. Test simulador

Este tipo de Test son un ejemplo de las distintas funciones del simulador y como ponerlas en práctica.

Buscar pelota roja

Añade un objeto pelota de color rojo al escenario. A continuación comienza la búsqueda del objeto realizando capturas del escenario y analizándolas mediante los umbrales seleccionados. Una vez detectado el objeto, el objetivo es acercarse hacia él una cierta distancia y pararse. Para ello calcula la distancia hacia él, si esta es menor o igual que la requerida, se queda parado; si no, se acerca un poco más y vuelve a calcularla, repitiendo este paso hasta que se encuentre a la distancia adecuada.



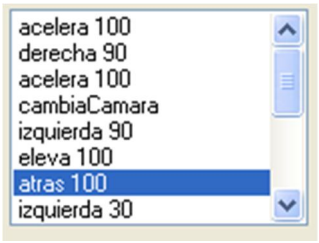
Ruta + Cámara Dinámica

Este test tiene la finalidad de demostrar el uso de la cámara dinámica para hacer simulaciones de rutas. Para ello carga una ruta predefinida y comienza la ejecución de las órdenes a la vez que la cámara se mueve siguiendo al robot para mantenerlo en pantalla.

Ruta + Cámara XML + Cámara Dinámica

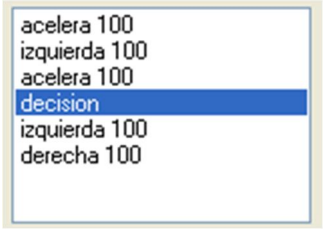
Este test es muy parecido al anterior, pero juega con las posibilidades de combinar la cámara XML, definida por el usuario, con la dinámica. Recordemos que la cámara XML permite al usuario definir los movimientos de cámara que desea que se produzcan durante la simulación y que, además, posee un comando cambia cámara que alterna esta con la dinámica.

En la imagen se puede observar la lista de comandos dada, con la orden "cambiaCámara" y otros comandos de cámara como "eleva 100".



Ruta + Decisión

Este caso carga una ruta que contiene un comando de decisión. Dicho comando lo único que hace es esperar a recibir una orden por medio del micro y una vez ejecutada, continúa con la ruta.

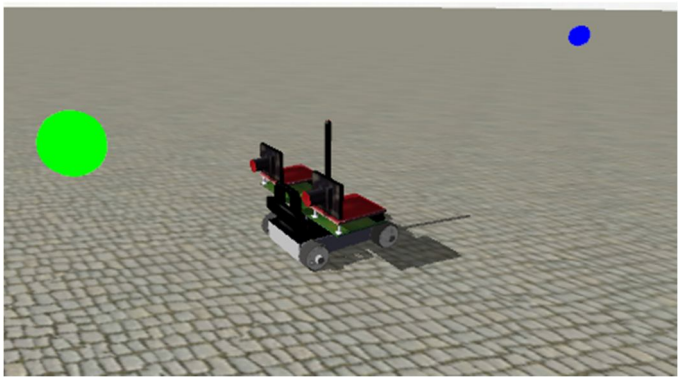


Ruta + Pelota

Carga una ruta con un comando de búsqueda. En este caso el comando "búsqueda rojo 100" buscará un objeto de color rojo y se acercará a él a una distancia de 100 cm. Una vez encontrado el objeto y situado el robot a 100 cm. continúa ejecutando los comandos que se encuentran después del comando búsqueda.

Ruta + pelotas tres colores

Genera 3 pelotas de colores azul, verde y rojo en el escenario. A continuación carga una ruta y comienza a ejecutarla. Esta ruta está compuesta de tres comandos de búsqueda cuyos parámetros indican que debe buscarse primero un objeto azul, luego uno rojo y por último uno verde.



5.2. Test reales

Esta colección de pruebas tiene la finalidad de probar el comportamiento del robot real para familiarizarse con su uso.

Cabe destacar que, antes de comenzar a ejecutarlos, es conveniente realizar las calibraciones necesarias para los algoritmos de visión descritas en anteriores apartados.

Además es necesaria tener establecida la conexión con la red inalámbrica del robot.

Ruta

Este caso tiene la finalidad de comprobar y observar la transmisión de órdenes a los motores del robot para que sea capaz de realizar la lista de comandos definidos por el usuario. Para ello carga una ruta predefinida del mismo modo que se hacía en el simulador. A continuación la manda ejecutar enviando las órdenes por Wi-fi al Surveyor.

Buscar Tres Pelotas

Este test es igual que el test del simulador en el que se buscaban tres objetos de diferentes colores pero en esta ocasión se ejecutará sobre el robot real utilizando las imágenes reales que transmiten las cámaras del Surveyor.

6. Herramientas utilizadas

6.1. Motor Newton

Newton Game Dinamics es una solución de código abierto para la simulación en tiempo real de entornos físicos. Viene integrado dentro de la API gráfica de desarrollo, TrueVision, de la que hablaremos más adelante.

Esta API se basa en el uso de las fuerzas físicas para realizar simulaciones. Proporciona un fácil manejo de la escena, comportamiento dinámico y detección de colisiones. Sobre todo centrándose en esta última característica, ya que se han optimizado los altos costes de los algoritmos de colisión y recorte o clipping. El motor implementa un programa de solución determinista, que no está basado en los tradicionales LCP o métodos iterativos, pero posee la estabilidad y la velocidad de ambos, respectivamente. Todo ello hace que ofrezca una respuesta bastante rápida en las simulaciones en tiempo real, lo que es ideal para nuestro proyecto.

Su integración es bastante sencilla y sin esta herramienta hubiéramos tenido que implementar grandes algoritmos para conseguir simular todo lo que hemos hecho si únicamente usamos una API de desarrollo como OpenGL o DirectX.

6.2. AForge

AForge.NET es un conjunto de librerías para C# sobre .NET diseñado para desarrolladores y programadores especializados en los campos de la visión artificial y la inteligencia computacional. Hacen uso de algoritmos de procesamiento de imágenes, redes neuronales o incluso algoritmos genéticos

Mediante estas librerías hemos sido capaces de obtener un grado óptimo de comunicación con el robot. Dichas comunicaciones se establecen gracias a una red Wi-Fi propia del SRV-1. Las imágenes son recibidas con una velocidad bastante aceptable, al igual que las órdenes enviadas desde nuestra aplicación al robot.

A pesar de que el envío de datos a los motores junto con la recepción constante por parte de las cámaras es todo lo buena que se requiere para una aplicación en tiempo real, nos gustaría destacar el papel que hemos jugado mezclando las distintas posibilidades de la librería para que la rapidez de la comunicación fuera óptima. Las imágenes de video están ajustadas a una calidad media, ahora bien, nuestro problema venía a la hora de obtener un fotograma del entorno para su posterior procesamiento. Esto requería una calidad bastante alta para poder tener una imagen nítida en la que no nos encontráramos con el problema del ruido. Pero si iniciábamos las cámaras con una calidad alta, el envío de datos y también su recepción no se producía adecuadamente o por lo menos no con la rapidez requerida por el programa. Nuestra solución se ha basado en un cambio dinámico

de la resolución, de tal manera que las cámaras envían a la aplicación con calidad media, pero en el momento en el que el robot desea obtener la fotografía, se produce un aumento de la resolución de la misma, restableciendo de nuevo a los valores medios una vez ha sido obtenida.

6.3. TrueVision

TrueVision es una API gráfica desarrollada en C++ sobre DirectX 9.0 que se puede utilizar con diversos lenguajes. En este proyecto el lenguaje utilizado será C#.

Esta API es capaz de simular las propiedades y características de distintos materiales (que se incluyen en la biblioteca). Posee un sistema que gestiona dinámicamente las luces sobre la escena, reproduciendo con gran exactitud la incidencia de la luz sobre los objetos o el terreno según su material.

Respecto a la creación y animación de las mallas, hay que resaltar que mediante llamadas internas al sistema podemos reproducir ciertos comportamientos predeterminados en las mallas. Gracias a esto el robot es simulado como un vehículo con cuatro ruedas y cuatro motores. Además el SDK contiene varios plug-ins y utilidades para el desarrollo software, como son el editor de shaders y la exportación de mallas desde varias herramientas de modelado 3D.

Las simulaciones de terreno se reproducen mediante mapas de intensidades en blanco y negro. Las zonas más claras corresponden a elevaciones de terreno, y las más oscuras a zonas de depresión.

Además integra el motor antes explicado para la simulación de colisiones y fuerzas físicas, lo que lo convierte en la opción más adecuada.

6.4. Plataforma .NET

La plataforma .NET ha sido desarrollada por Microsoft. Es un componente de software que provee un extenso conjunto de soluciones predefinidas con especial énfasis en la independencia de la plataforma de hardware, transparencia de redes, que facilita un rápido desarrollo de aplicaciones. Pretende reunir las ventajas de lenguajes como C, C++ y Visual Basic.

La herramienta de desarrollo trabaja del siguiente modo. Primero compila el código fuente en un código intermedio, el CIL *(*Common Intermediate Language*)*, similar al BYTECODE de Java. Para generarlo, el compilador se basa en un estándar de especificación CLS *(*Common Language Specification*)* que determina las reglas necesarias para crear el código MSIL compatible con el CLR. El segundo paso se basa en un compilador JIT *(*Just-In-

Time*)* que genera el código máquina real que se ejecuta en la plataforma del cliente. De esta forma se consigue con .NET independencia de la plataforma de hardware.

Por último señalar, que el entorno de utilizado ha sido Visual Studio 2008. Es un entorno de desarrollo integrado para aplicaciones Windows, que soporta lenguajes como C++, C#, Visual J#, ASP:NET y Visual Basic .NET.

7. Conclusiones

Desde el principio nuestro objetivo fue ampliar la funcionalidad del robot Surveyor, creando una aplicación que simplifique su manejo y extienda sus capacidades.

Hemos conseguido desarrollar un software capaz de ser manejado por usuarios ajenos al mundo de la informática o la robótica, explotando al mayor nivel posible las ventajas de su manejo por Wi-Fi, su capacidad de maniobrabilidad y las posibilidades que nos brindan sus dos cámaras.

El proyecto es el resultado de juntar tres disciplinas de la informática, como son: la informática gráfica, la robótica y la inteligencia artificial. Nuestros conocimientos en dichas materias eran básicos, así que en este proyecto se aúna el esfuerzo de todo el equipo durante el año por profundizar en estos campos y a su vez nos ha supuesto un gran reto y una gran satisfacción poder trabajar en el desarrollo de una aplicación en tiempo real, con todas las dificultades que eso supone: respuesta rápida del sistema, ejecución de órdenes en un intervalo de tiempo aceptable, recepción de datos en tiempo real, etc…

Las aplicaciones de esta herramienta son bastante amplias. Esto es debido a que no nos hemos centrado en desarrollar una utilidad concreta con el robot, sino que nos hemos concentrado en generar una serie de funcionalidades que puedan ser utilizadas en diversos campos, sin ir más lejos, es de gran utilidad en el mundo de la navegación no tripulada o la exploración mediante rutas.

Para terminar, nos gustaría destacar el apoyo recibido por parte del profesor, como nos ha dirigido durante el año para conseguir los objetivos que nos planteamos al abordar este proyecto y cómo nos ha motivado para no conformarnos con lo conseguido, proponiéndonos la resolución de nuevos retos cuando las metas iniciales estaban finalizadas.

8. Bibliografía

Robot Surveyor:

http://www.surveyor.com/SRV_info.html

Librería para controlar el robot con el lenguaje objeto:

www.aforgenet.com

API para el control de voz:

http://en.wikipedia.org/wiki/Microsoft_Speech_API

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=5e86ec97-40a7-453f-b0ee-6583171b4530&displaylang=en

Distintos ejemplos de C# (para integrar APIs, lectura de XML,...):

http://www.codeproject.com

http://www.c-sharpcorner.com/

Página de TrueVision y foros para consultar información:

www.truevision3d.com

MSDN de Microsoft:

http://msdn.microsoft.com/es-es/library/ms123401